

FIGURE 1

[illegible]

FIGURE 2

MSSQPAGNQTS PGATEDYSYGSWYIDEPQGGEELQPEGEVPSCHTSIPPGLYHACLASLS
 ILVLLLLLAMLVRRRQLWPD CVRGRPGLSPVD FLAGDRPRAVPAAVFMVLLSSLCLLLPD
 EDALPFLTLASAPSQDGKTEAPRGAWKILGLFYAALYYPLAACATAGHTAAHLLGSTLS
 5 WAHLGVQVWQRAECPQVPKIYKYYSLLASLPLLLGLGFLSLWYPVQLVRSFSRRTGAGSK
 GLQSSYSEEYLRNLLCRKKLGSSYHTSKHGFLSWARVCLRHCIYTPQPGFHLPLKLVLSA
 TLGTAIYQVALLLLVGVPPTIQKVRAGVTTDVSYLLAGFGIVLSEDKQEVVELVKHHLW
 ALEVCYISALVLSCLLTFLVLMRSLVTHRTNLRALHRGAALDLSPLHRSPHPSRQAI FCW
 MSFSAYQTAFICLGLLVQQIIFFLGTTALAFLVLMPLVHGRNLLLFRSLESSWPFWLT LA
 10 LAVILQNMAAHWVFLETHDGH PQLTNRRLVLYAATFLLFPLNVLVGAMVATWRVLLSALYN
 AIHLGQMDLSLLPPRAATLDPGYYTYRNFLKIEVSQSH PAMTAFCSLLLQAQSLLPRTMA
 APQDSL RPGEED EGMQLLQTKDSMAKGARPGASRGRARWGLAYTLLHNPTLQVFRKTALL
 GANGAQP

Important features of the protein:

Signal peptide:

None

Transmembrane domain:

54-69
 102-119
 148-166
 207-222
 301-320
 364-380
 431-451
 474-489
 560-535

Motif file:

Motif name: N-glycosylation site.

8-12

Motif name: N-myristoylation site.

50-56
 176-182
 241-247
 317-323
 341-347
 525-531
 627-633
 631-637
 640-646
 661-667

Motif name: Prokaryotic membrane lipoprotein lipid attachment site.

364-375

Motif name: ATP/GTP-binding site motif A (P-loop).

132-140

FIGURE 3A

PRO XXXXXXXXXXXXXXXXX (Length = 15 amino acids)

Comparison Protein XXXXXYYYYYYYY (Length = 12 amino acids)

5

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide) =

10

5 divided by 15 = 33.3%

XXXXXXXXXXXXXXXXXXXX

FIGURE 3B

PRO	XXXXXXXXXX	(Length = 10 amino acids)
Comparison Protein	XXXXXXYYYYYYZZYZ	(Length = 15 amino acids)

5

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide) =

10

5 divided by 10 = 50%

XXXXXXXXXX
XXXXXXYYYYYYZZYZ

FIGURE 3C

PRO-DNA NNNNNNNNNNNNNNNN (Length = 14
nucleotides)

5 Comparison DNA NNNNNNLLLLLLLLLL (Length = 16
nucleotides)

% nucleic acid sequence identity =

10 (the number of identically matching nucleotides between the two nucleic acid sequences as
determined by ALIGN-2) divided by (the total number of nucleotides of the PRO-DNA
nucleic acid sequence) =

6 divided by 14 = 42.9%

15

Downloaded from www.ascp.com

[illegible]

5

10

4 divided by 12 = 33 3%

FIGURE 4A

```

/*
 *
 * C-C increased from 12 to 15
 * Z is average of EQ
 * B is average of ND
 * match with stop is _M; stop-stop = 0; J (joker) match = 0
 */
#define _M      -8      /* value of a match with a stop */

int      _day[26][26] = {
/*      A B C D E F G H I J K L M N O P Q R S T U V W X Y Z */
/* A */      { 2, 0, -2, 0, 0, -4, 1, -1, -1, 0, -1, -2, -1, 0, _M, 1, 0, -2, 1, 1, 0, 0, -6, 0, -3, 0},
/* B */      { 0, 3, -4, 3, 2, -5, 0, 1, -2, 0, 0, -3, -2, 2, _M, -1, 1, 0, 0, 0, 0, -2, -5, 0, -3, 1},
/* C */      {-2, -4, 15, -5, -5, -4, -3, -3, -2, 0, -5, -6, -5, -4, _M, -3, -5, -4, 0, -2, 0, -2, -8, 0, 0, -5},
/* D */      { 0, 3, -5, 4, 3, -6, 1, 1, -2, 0, 0, -4, -3, 2, _M, -1, 2, -1, 0, 0, 0, -2, -7, 0, -4, 2},
/* E */      { 0, 2, -5, 3, 4, -5, 0, 1, -2, 0, 0, -3, -2, 1, _M, -1, 2, -1, 0, 0, 0, -2, -7, 0, -4, 3},
/* F */      {-4, -5, -4, -6, -5, 9, -5, -2, 1, 0, -5, 2, 0, -4, _M, -5, -5, -4, -3, -3, 0, -1, 0, 0, 7, -5},
/* G */      { 1, 0, -3, 1, 0, -5, 5, -2, -3, 0, -2, -4, -3, 0, _M, -1, -1, -3, 1, 0, 0, -1, -7, 0, -5, 0},
/* H */      {-1, 1, -3, 1, 1, -2, -2, 6, -2, 0, 0, -2, -2, 2, _M, 0, 3, 2, -1, -1, 0, -2, -3, 0, 0, 2},
/* I */      {-1, -2, -2, -2, -2, 1, -3, -2, 5, 0, -2, 2, 2, -2, _M, -2, -2, -2, -1, 0, 0, 4, -5, 0, -1, -2},
/* J */      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, _M, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
/* K */      {-1, 0, -5, 0, 0, -5, -2, 0, -2, 0, 5, -3, 0, 1, _M, -1, 1, 3, 0, 0, 0, -2, -3, 0, -4, 0},
/* L */      {-2, -3, -6, -4, -3, 2, -4, -2, 2, 0, -3, 6, 4, -3, _M, -3, -2, -3, -3, -1, 0, 2, -2, 0, -1, -2},
/* M */      {-1, -2, -5, -3, -2, 0, -3, -2, 2, 0, 0, 4, 6, -2, _M, -2, -1, 0, -2, -1, 0, 2, -4, 0, -2, -1},
/* N */      { 0, 2, -4, 2, 1, -4, 0, 2, -2, 0, 1, -3, 2, 2, _M, -1, 1, 0, 1, 0, 0, -2, -4, 0, -2, 1},
/* O */      { _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M},
0, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M, _M},
/* P */      { 1, -1, -3, -1, -1, -5, -1, 0, -2, 0, -1, -3, -2, -1, _M, 6, 0, 0, 1, 0, 0, -1, -6, 0, -5, 0},
/* Q */      { 0, 1, -5, 2, 2, -5, -1, 3, -2, 0, 1, -2, -1, 1, _M, 0, 4, 1, -1, -1, 0, -2, -5, 0, -4, 3},
/* R */      {-2, 0, -4, -1, -1, -4, -3, 2, -2, 0, 3, -3, 0, 0, _M, 0, 1, 6, 0, -1, 0, -2, 2, 0, -4, 0},
/* S */      { 1, 0, 0, 0, 0, -3, 1, -1, -1, 0, 0, -3, -2, 1, _M, 1, -1, 0, 2, 1, 0, -1, -2, 0, -3, 0},
/* T */      { 1, 0, -2, 0, 0, -3, 0, -1, 0, 0, 0, -1, -1, 0, _M, 0, -1, -1, 1, 3, 0, 0, -5, 0, -3, 0},
/* U */      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, _M, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
/* V */      { 0, -2, -2, -2, -2, -1, -1, -2, 4, 0, -2, 2, 2, -2, _M, -1, -2, -2, -1, 0, 0, 4, -6, 0, -2, -2},
/* W */      {-6, -5, -8, -7, -7, 0, -7, -3, -5, 0, -3, -2, -4, -4, _M, -6, -5, 2, -2, -5, 0, -6, 17, 0, 0, -6},
/* X */      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, _M, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
/* Y */      {-3, -3, 0, -4, -4, 7, -5, 0, -1, 0, -4, -1, -2, -2, _M, -5, -4, -4, -3, -3, 0, -2, 0, 0, 10, -4},
/* Z */      { 0, 1, -5, 2, 3, -5, 0, 2, -2, 0, 0, -2, -1, 1, _M, 0, 3, 0, 0, 0, 0, -2, -6, 0, -4, 4}
};

```

FIGURE 4B

```

/*
*/
#include <stdio.h>
5  #include <ctype.h>

#define MAXJMP      16      /* max jumps in a diag */
#define MAXGAP      24      /* don't continue to penalize gaps larger than this */
#define JMPS        1024    /* max jmps in an path */
10  #define MX        4      /* save if there's at least MX-1 bases since last jmp */

#define DMAT         3      /* value of matching bases */
#define DMIS         0      /* penalty for mismatched bases */
15  #define DINS0      8      /* penalty for a gap */
#define DINS1        1      /* penalty per base */
#define PINS0        8      /* penalty for a gap */
#define PINS1        4      /* penalty per residue */

struct jmp {
20     short          n[MAXJMP];    /* size of jmp (neg for dely) */
     unsigned short  x[MAXJMP];    /* base no of jmp in seq x */
};                                  /* limits seq to 216-1 */

struct diag {
25     int            score,         /* score at last jmp */
     long           offset;        /* offset of prev block */
     short          ijmp;          /* current jmp index */
     struct jmp      jp;           /* list of jmps */
};

30  struct path {
     int            spc;            /* number of leading spaces */
     short          n[JMPS];       /* size of jmp (gap) */
35     int           x[JMPS];       /* loc of jmp (last elem before gap) */
};

char          *ofile;              /* output file name */
char          *namex[2];           /* seq names getseqs() */
40  char          *prog;            /* prog name for err msgs */
char          *seqx[2];            /* seqs getseqs() */
int           dmax;                /* best diag: nw() */
int           dmax0;              /* final diag */
int           dna;                /* set if dna: main() */
45  int           endgaps;          /* set if penalizing end gaps */
int           gapx, gapy;          /* total gaps in seqs */
int           len0, len1;          /* seq lens */
int           ngapx, ngapy;        /* total size of gaps */
int           smax;               /* max score nw() */
int           *xbm;               /* bitmap for matching */
50  long          offset;           /* current offset in jmp file */
     struct      diag             /* holds diagonals */
     struct      path             /* holds path for seqs */
pp[2];

55  char          *calloc(), *malloc(), *index(), *strcpy();
char          *getseq(), *g_calloc();

```


FIGURE 4C

```
/* Needleman-Wunsch alignment program
```

```
*
```

```
* usage: progs file1 file2
```

```
* where file1 and file2 are two dna or two protein sequences.
```

```
* The sequences can be in upper- or lower-case and may contain ambiguity
```

```
* Any lines beginning with ';', '>' or '<' are ignored
```

```
* Max file length is 65535 (limited by unsigned short x in the jmp struct)
```

```
* A sequence with 1/3 or more of its elements ACGTU is assumed to be DNA
```

```
* Output is in the file "align.out"
```

```
*
```

```
* The program may create a tmp file in /tmp to hold info about traceback.
```

```
* Original version developed under BSD 4.3 on a vax 8650
```

```
*/
```

```
#include "nw.h"
```

```
#include "day.h"
```

```
static _dbval[26] = {
```

```
1,14,2,13,0,0,4,11,0,0,12,0,3,15,0,0,0,5,6,8,8,7,9,0,10,0
```

```
};
```

```
static _pbval[26] = {
```

```
1, 2|(1<<('D'-'A'))|(1<<('N'-'A')), 4, 8, 16, 32, 64,
```

```
128, 256, 0xFFFFFFFF, 1<<10, 1<<11, 1<<12, 1<<13, 1<<14,
```

```
1<<15, 1<<16, 1<<17, 1<<18, 1<<19, 1<<20, 1<<21, 1<<22,
```

```
1<<23, 1<<24, 1<<25|(1<<('E'-'A'))|(1<<('Q'-'A'))
```

```
};
```

```
main(ac, av)
```

```
int ac;
```

```
char *av[],
```

```
{
```

```
prog = av[0];
```

```
if (ac != 3) {
```

```
fprintf(stderr, "usage: %s file1 file2\n", prog);
```

```
fprintf(stderr, "where file1 and file2 are two dna or two protein sequences\n");
```

```
fprintf(stderr, "The sequences can be in upper- or lower-case\n");
```

```
fprintf(stderr, "Any lines beginning with ';', '>' or '<' are ignored\n");
```

```
fprintf(stderr, "Output is in the file \"align.out\"\n");
```

```
exit(1);
```

```
}
```

```
namex[0] = av[1];
```

```
namex[1] = av[2];
```

```
seqx[0] = getseq(namex[0], &len0);
```

```
seqx[1] = getseq(namex[1], &len1);
```

```
xbm = (dna)? _dbval : _pbval;
```

```
endgaps = 0;
```

```
/* 1 to penalize endgaps */
```

```
ofile = "align.out";
```

```
/* output file */
```

```
nw();
```

```
/* fill in the matrix, get the possible jumps */
```

```
readjumps();
```

```
/* get the actual jumps */
```

```
print();
```

```
/* print stats, alignment */
```

```
cleanup(0);
```

```
/* unlink any tmp files */
```

```
}
```

main

FIGURE 4D

```

/* do the alignment, return best score: main()
 * dna: values in Fitch and Smith, PNAS, 80, 1382-1386, 1983
 * pro: PAM 250 values
5  * When scores are equal, we prefer mismatches to any gap, prefer
 * a new gap to extending an ongoing gap, and prefer a gap in seqx
 * to a gap in seq y.
 */

```

```

nw()

```

nw

```

10 {
    char      *px, *py;      /* seqs and ptrs */
    int       *ndely, *dely;  /* keep track of dely */
    int       ndelx, delx;    /* keep track of delx */
    int       *tmp;          /* for swapping row0, row1 */
15  int       mis;           /* score for each type */
    int       ins0, ins1,     /* insertion penalties */
    register  id;            /* diagonal index */
    register  ij;           /* jmp index */
    register  *col0, *col1;   /* score for curr, last row */
20  register  xx, yy;        /* index into seqs */

    dx = (struct diag *)g_calloc("to get diag", len0 + len1 + 1, sizeof(struct diag));

    ndely = (int *)g_calloc("to get ndely", len1 + 1, sizeof(int));
    dely = (int *)g_calloc("to get dely", len1 + 1, sizeof(int));
    col0 = (int *)g_calloc("to get col0", len1 + 1, sizeof(int));
    col1 = (int *)g_calloc("to get col1", len1 + 1, sizeof(int));
    ins0 = (dna)? DINS0 : PINS0;
    ins1 = (dna)? DINS1 : PINS1;

    smax = -10000;
    if (endgaps) {
        for (col0[0] = dely[0] = -ins0, yy = 1, yy <= len1; yy++) {
            col0[yy] = dely[yy] = col0[yy-1] - ins1;
            ndely[yy] = yy;
        }
        col0[0] = 0;      /* Waterman Bull Math Biol 84 */
    }
    else
        for (yy = 1; yy <= len1; yy++)
            dely[yy] = -ins0;

    /* fill in match matrix
    */
45  for (px = seqx[0], xx = 1; xx <= len0; px++, xx++) {
        /* initialize first entry in col
        */
        if (endgaps) {
            if (xx == 1)
                col1[0] = delx = -(ins0 + ins1);
            else
                col1[0] = delx = col0[0] - ins1;
            ndelx = xx;
        }
55  else {
            col1[0] = 0;
            delx = -ins0;
            ndelx = 0;
        }
60  }

```

FIGURE 4E

...nw

```

for (py = seqx[1], yy = 1; yy <= len1; py++, yy++) {
    mis = col0[yy-1];
    if (dna)
        mis += (xbm[*px-'A']&xbm[*py-'A'])? DMAT : DMIS;
    else
        mis += _day[*px-'A'][*py-'A'];

    /* update penalty for del in x seq;
     * favor new del over ongoing del
     * ignore MAXGAP if weighting endgaps
     */
    if (endgaps || ndely[yy] < MAXGAP) {
        if (col0[yy] - ins0 >= dely[yy]) {
            dely[yy] = col0[yy] - (ins0+ins1);
            ndely[yy] = 1;
        } else {
            dely[yy] -= ins1;
            ndely[yy]++;
        }
    } else {
        if (col0[yy] - (ins0+ins1) >= dely[yy]) {
            dely[yy] = col0[yy] - (ins0+ins1);
            ndely[yy] = 1;
        } else
            ndely[yy]++;
    }

    /* update penalty for del in y seq;
     * favor new del over ongoing del
     */
    if (endgaps || ndelx < MAXGAP) {
        if (col1[yy-1] - ins0 >= delx) {
            delx = col1[yy-1] - (ins0+ins1);
            ndelx = 1;
        } else {
            delx -= ins1;
            ndelx++;
        }
    } else {
        if (col1[yy-1] - (ins0+ins1) >= delx) {
            delx = col1[yy-1] - (ins0+ins1);
            ndelx = 1;
        } else
            ndelx++;
    }

    /* pick the maximum score; we're favoring
     * mis over any del and delx over dely
     */

```

FIGURE 4F

...nw

```

id = xx - yy + len1 - 1;
if (mis >= delx && mis >= dely[yy])
    coll[yy] = mis;
else if (delx >= dely[yy]) {
    coll[yy] = delx;
    ij = dx[id].ijmp;
    if (dx[id].jp.n[0] && (!dna || (ndelx >= MAXJMP
    && xx > dx[id].jp.x[ij]+MX) || mis > dx[id].score+DINS0)) {
        dx[id].ijmp++;
        if (++ij >= MAXJMP) {
            writejumps(id);
            ij = dx[id].ijmp = 0;
            dx[id].offset = offset;
            offset += sizeof(struct jmp) + sizeof(offset);
        }
        dx[id].jp.n[ij] = ndelx;
        dx[id].jp.x[ij] = xx;
        dx[id].score = delx;
    }
    else {
        coll[yy] = dely[yy];
        ij = dx[id].ijmp;

if (dx[id].jp.n[0] && (!dna || (ndely[yy] >= MAXJMP
    && xx > dx[id].jp.x[ij]+MX) || mis > dx[id].score+DINS0)) {
        dx[id].ijmp++;
        if (++ij >= MAXJMP) {
            writejumps(id);
            ij = dx[id].ijmp = 0;
            dx[id].offset = offset;
            offset += sizeof(struct jmp) + sizeof(offset);
        }
        dx[id].jp.n[ij] = ndely[yy];
        dx[id].jp.x[ij] = xx;
        dx[id].score = dely[yy];
    }
    if (xx == len0 && yy < len1) {
        /* last col
        */
        if (endgaps)
            coll[yy] -= ins0+ins1*(len1-yy);
        if (coll[yy] > smax) {
            smax = coll[yy];
            dmax = id;
        }
    }
}
if (endgaps && xx < len0)
    coll[yy-1] -= ins0+ins1*(len0-xx);
if (coll[yy-1] > smax) {
    smax = coll[yy-1];
    dmax = id;
}
tmp = col0; col0 = coll; coll = tmp;
}
(void) free((char *)ndely);
(void) free((char *)dely);
(void) free((char *)col0);(void) free((char *)coll);}

```

FIGURE 4G

```

/*
 *
 * print() -- only routine visible outside this module
 *
 * static:
 * getmat() -- trace back best path, count matches: print()
 * pr_align() -- print alignment of described in array p[]: print()
 * dumpblock() -- dump a block of lines with numbers, stars: pr_align()
 * nums() -- put out a number line: dumpblock()
 * putline() -- put out a line (name, [num], seq, [num]): dumpblock()
 * stars() -- put a line of stars: dumpblock()
 * stripname() -- strip any path and prefix from a seqname
 */

#include "nw h"

#define SPC      3
#define P_LINE  256    /* maximum output line */
#define P_SPC    3      /* space between name or num and seq */

extern  _day[26][26];
int     olen;           /* set output line length */
FILE    *fx,            /* output file */

print()
{
    int     lx, ly, firstgap, lastgap;    /* overlap */

    if ((fx = fopen(ofile, "w")) == 0) {
        fprintf(stderr, "%s can't write %s\n", prog, ofile);
        cleanup(1);
    }
    fprintf(fx, "< first sequence: %s (length = %d)\n", namex[0], len0);
    fprintf(fx, "< second sequence: %s (length = %d)\n", namex[1], len1);
    olen = 60;
    lx = len0;
    ly = len1;
    firstgap = lastgap = 0;
    if (dmax < len1 - 1) {    /* leading gap in x */
        pp[0].spc = firstgap = len1 - dmax - 1;
        ly -= pp[0].spc;
    }
    else if (dmax > len1 - 1) { /* leading gap in y */
        pp[1].spc = firstgap = dmax - (len1 - 1);
        lx -= pp[1].spc;
    }
    if (dmax0 < len0 - 1) {    /* trailing gap in x */
        lastgap = len0 - dmax0 - 1;
        lx -= lastgap;
    }
    else if (dmax0 > len0 - 1) { /* trailing gap in y */
        lastgap = dmax0 - (len0 - 1);
        ly -= lastgap;
    }
    getmat(lx, ly, firstgap, lastgap);
    pr_align();
}

```

print

FIGURE 4H

```

/*
 * trace back the best path, count matches
 */
5 static
getmat(lx, ly, firstgap, lastgap)                                getmat
    int      lx, ly;                /* "core" (minus endgaps) */
    int      firstgap, lastgap;     /* leading trailing overlap */
{
10     int      nm, i0, i1, siz0, siz1;
    char      outx[32];
    double    pct;
    register  n0, n1;
    register char *p0, *p1;

15     /* get total matches, score
    */
    i0 = i1 = siz0 = siz1 = 0;
    p0 = seqx[0] + pp[1] spc;
    p1 = seqx[1] + pp[0] spc;
    n0 = pp[1].spc + 1;
    n1 = pp[0].spc + 1;

20     nm = 0;
    while ( *p0 && *p1 ) {
        if (siz0) {
            p1 ++;
            n1 ++;
            siz0--;
25         }
        else if (siz1) {
            p0 ++;
            n0 ++;
            siz1--;
30         }
        else {
            if (xbm[*p0-'A']&xbm[*p1-'A'])
                nm ++;
            if (n0++ == pp[0].x[i0])
                siz0 = pp[0].n[i0++];
            if (n1++ == pp[1].x[i1])
                siz1 = pp[1].n[i1++];
35             p0 ++;
            p1 ++;
40         }
45     }

    /* pct homology:
    * if penalizing endgaps, base is the shorter seq
    * else, knock off overhangs and take shorter core
    */
    if (endgaps)
        lx = (len0 < len1)? len0 : len1;
    else
50         lx = (lx < ly)? lx : ly;
    pct = 100.*(double)nm/(double)lx;
    fprintf(fx, "\n");
    fprintf(fx, "< %d match%s in an overlap of %d: %.2f percent similarity\n",
60         nm, (nm == 1)? "" : "es", lx, pct);

```

FIGURE 4I

```

fprintf(fx, "< gaps in first sequence: %d", gapx);
if (gapx) {
    (void) sprintf(outx, " (%d %s%s)",
        ngapx, (dna)? "base": "residue", (ngapx == 1)? "": "s");
    fprintf(fx, "%s", outx);

    fprintf(fx, ", gaps in second sequence: %d", gapy);
    if (gapy) {
        (void) sprintf(outx, " (%d %s%s)",
            ngapy, (dna)? "base": "residue", (ngapy == 1)? "": "s");
        fprintf(fx, "%s", outx);
    }
    if (dna)
        fprintf(fx,
            "\n< score: %d (match = %d, mismatch = %d, gap penalty = %d + %d per base)\n",
            smax, DMAT, DMIS, DINS0, DINS1);
    else
        fprintf(fx,
            "\n< score: %d (Dayhoff PAM 250 matrix, gap penalty = %d + %d per residue)\n",
            smax, PINS0, PINS1);
    if (endgaps)
        fprintf(fx,
            "< endgaps penalized left endgap: %d %s%s, right endgap: %d %s%s\n",
            firstgap, (dna)? "base": "residue", (firstgap == 1)? "": "s",
            lastgap, (dna)? "base": "residue", (lastgap == 1)? "": "s");
    else
        fprintf(fx, "< endgaps not penalized\n");
}

static nm;          /* matches in core -- for checking */
static lmax;        /* lengths of stripped file names */
static ij[2];       /* jmp index for a path */
static nc[2];       /* number at start of current line */
static ni[2];       /* current elem number -- for gapping */
static siz[2];
static char *ps[2];  /* ptr to current element */
static char *po[2];  /* ptr to next output char slot */
static char out[2][P_LINE]; /* output line */
static char star[P_LINE]; /* set by stars() */

/*
 * print alignment of described in struct path pp[]
 */
static
pr_align()
{
    int nn;          /* char count */
    int more;
    register i;

    for (i = 0, lmax = 0; i < 2; i++) {
        nn = stripname(namex[i]);
        if (nn > lmax)
            lmax = nn;

        nc[i] = 1;
        ni[i] = 1;
        siz[i] = ij[i] = 0;
        ps[i] = seqx[i];
        po[i] = out[i];
    }
}

```

...getmat

pr_align

FIGURE 4J

```

5      for (nn = nm = 0, more = 1; more; ) {
        for (i = more = 0; i < 2; i++) {
            /*
            * do we have more of this sequence?
            */
            if (!*ps[i])
                continue;

            more++;

            if (pp[i].spc) { /* leading space */
                *po[i]++ = ' ';
                pp[i].spc--;
            }
            else if (siz[i]) { /* in a gap */
                *po[i]++ = '-';
                siz[i]--;
            }
            else { /* we're putting a seq element
            */
                *po[i] = *ps[i];
                if (islower(*ps[i]))
                    *ps[i] = toupper(*ps[i]);
                po[i]++;
                ps[i]++;

                /*
                * are we at next gap for this seq?
                */
                if (ni[i] == pp[i].x[ij[i]]) {
                    /*
                    * we need to merge all gaps
                    * at this location
                    */
                    siz[i] = pp[i].n[ij[i]] + +;
                    while (ni[i] == pp[i].x[ij[i]])
                        siz[i] += pp[i].n[ij[i]] + +;
                }
                ni[i]++;
            }
        }
        if (++nn == olen || !more && nm) {
            dumpblock();
            for (i = 0; i < 2; i++)
                po[i] = out[i];
            nm = 0;
        }
    }

    /*
    * dump a block of lines, including numbers, stars: pr_align()
    */
    static
    dumpblock()
    {
        register i;

        for (i = 0; i < 2; i++)
            *po[i]-- = '\0';
    }
}

...pr_align

dumpblock

60

```


FIGURE 4K

...dumpblock

```
(void) putc('\n', fx);
5   for (i = 0; i < 2; i++) {
        if (*out[i] && (*out[i] != ' ' || *(po[i]) != ' ')) {
            if (i == 0)
                nums(i);
            if (i == 0 && *out[1])
10          stars();
            putline(i);
            if (i == 0 && *out[1])
                fprintf(fx, star);
            if (i == 1)
                nums(i);
        }
    }
}
```

```
20 /*
 * put out a number line: dumpblock()
 */
static
nums(ix)
25 {
    int      ix,      /* index in out[] holding seq line */
    char      nline[P_LINE],
    register  i, j;
    register char *pn, *px, *py;

    for (pn = nline, i = 0; i < lmax + P_SPC; i++, pn++)
        *pn = ' ';
    for (i = nc[ix], py = out[ix]; *py; py++, pn++) {
        if (*py == ' ' || *py == '-')
            *pn = ' ';
        else {
            if (i%10 == 0 || (i == 1 && nc[ix] != 1)) {
                j = (i < 0)? -i : i;
                for (px = pn; j /= 10, px--)
                    *px = j%10 + '0';
                if (i < 0)
                    *px = '-';
            }
            else
                *pn = ' ';
            i++;
        }
    }
    *pn = '\0';
    nc[ix] = i;
    for (pn = nline; *pn; pn++)
        (void) putc(*pn, fx);
    (void) putc('\n', fx);
}
```

```
55 /*
 * put out a line (name, [num], seq, [num]): dumpblock()
 */
static
60 putline(ix)
    int      ix;
{
```

nums

putline

FIGURE 4L

...putline

```

5      int          i;
      register char *px;

      for (px = namex[ix], i = 0; *px && *px != ':'; px++, i++)
          (void) putc(*px, fx);
10     for (; i < lmax+P_SPC; i++)
          (void) putc(' ', fx);

      /* these count from 1:
       * m[] is current element (from 1)
       * nc[] is number at start of current line
       */
15     for (px = out[ix]; *px; px++)
          (void) putc(*px&0x7F, fx);
      (void) putc('\n', fx);
20 }

```

```

/*
 * put a line of stars (seqs always in out[0], out[1]) dumpblock()
 */

```

```

25 static
stars()

```

stars

```

{
    int          i;
    register char *p0, *p1, cx, *px;

    if (!*out[0] || (*out[0] == ' ' && *(p0[0]) == ' ') ||
        !*out[1] || (*out[1] == ' ' && *(p0[1]) == ' '))
        return;
    px = star;
35     for (i = lmax+P_SPC; i; i--)
        *px++ = ' ';

    for (p0 = out[0], p1 = out[1]; *p0 && *p1; p0++, p1++) {
40         if (isalpha(*p0) && isalpha(*p1)) {

            if (xbm[*p0-'A']&xbm[*p1-'A']) {
                cx = '*';
                nm++;
            }
45             else if (!dna && _day[*p0-'A'][*p1-'A'] > 0)
                cx = '.';
            else
                cx = ' ';

50         }
        else
            cx = ' ';
        *px++ = cx;
55     }
    *px++ = '\n';
    *px = '\0';
}

```

FIGURE 4M

```
/*
 * strip path or prefix from pn, return len: pr_align()
 */
static
stripname(pn)
    char    *pn;    /* file name (may be path) */
{
    register char    *px, *py;

    py = 0;
    for (px = pn; *px; px++)
        if (*px == '/')
            py = px + 1;
    if (py)
        (void) strcpy(pn, py);
    return(strlen(pn));
}
```

stripname

FIGURE 40

...getseq

```

py = pseq + 4;
*len = tlen;
rewind(fp),

while (fgets(line, 1024, fp)) {
    if (*line == ';' || *line == '<' || *line == '>')
        continue;
    for (px = line; *px != '\n'; px++) {
        if (isupper(*px))
            *py++ = *px;
        else if (islower(*px))
            *py++ = toupper(*px);
        if (index("ATGCU",*(py-1)))
            natgc++;
    }
    *py++ = '\0';
    *py = '\0';
    (void) fclose(fp);
    dna = natgc > (tlen/3);
    return(pseq + 4),
}

char *
g_calloc(msg, nx, sz)
char *msg, /* program, calling routine */
int nx, sz, /* number and size of elements */
{
    char *px, *calloc();

    if ((px = calloc((unsigned)nx, (unsigned)sz)) == 0) {
        if (*msg) {
            fprintf(stderr, "%s: g_calloc() failed %s (n=%d, sz=%d)\n", prog, msg, nx, sz);
            exit(1);
        }
    }
    return(px),
}

/*
 * get final jmps from dx[] or tmp file, set pp[], reset dmax: main()
 */
readjmps()
{
    int fd = -1;
    int siz, i0, i1;
    register i, j, xx;

    if (fj) {
        (void) fclose(fj);
        if ((fd = open(jname, O_RDONLY, 0)) < 0) {
            fprintf(stderr, "%s: can't open() %s\n", prog, jname);
            cleanup(1);
        }
    }
    for (i = i0 = i1 = 0, dmax0 = dmax, xx = len0; i++) {
        while (1) {
            for (j = dx[dmax].ijmp; j >= 0 && dx[dmax].jp x[j] >= xx; j--)
                ;

```

FIGURE 4P

...readjumps

```

5      if (j < 0 && dx[dmax].offset && fj) {
        (void) lseek(fd, dx[dmax].offset, 0);
        (void) read(fd, (char *)&dx[dmax].jp, sizeof(struct jmp));
        (void) read(fd, (char *)&dx[dmax].offset, sizeof(dx[dmax].offset));
        dx[dmax].ijmp = MAXJMP-1;
      }
10     else
        break;
    }
    if (i >= JMPS) {
        fprintf(stderr, "%s: too many gaps in alignment\n", prog);
        cleanup(1);
    }
15     if (j >= 0) {
        siz = dx[dmax].jp.n[j];
        xx = dx[dmax].jp.x[j];
        dmax += siz;
        if (siz < 0) { /* gap in second seq */
            pp[1].n[i1] = -siz;
            xx += siz,

            /* id = xx - yy + len1 - 1
            */
            pp[1].x[i1] = xx - dmax + len1 - 1;
            gapy ++,
            ngapy -= siz;
        /* ignore MAXGAP when doing endgaps */
            siz = (-siz < MAXGAP || endgaps)? -siz : MAXGAP;
            i1 ++,
        }
        else if (siz > 0) { /* gap in first seq */
            pp[0].n[i0] = siz;
            pp[0].x[i0] = xx;
            gapx ++;
            ngapx += siz;
        /* ignore MAXGAP when doing endgaps */
            siz = (siz < MAXGAP || endgaps)? siz : MAXGAP;
            i0 ++;
        }
    }
    else
        break;
45 }

/* reverse the order of jumps
*/
50 for (j = 0, i0--; j < i0; j++, i0--) {
    i = pp[0].n[j]; pp[0].n[j] = pp[0].n[i0]; pp[0].n[i0] = i;
    i = pp[0].x[j]; pp[0].x[j] = pp[0].x[i0]; pp[0].x[i0] = i;
}
for (j = 0, i1--; j < i1; j++, i1--) {
55     i = pp[1].n[j]; pp[1].n[j] = pp[1].n[i1]; pp[1].n[i1] = i;
    i = pp[1].x[j]; pp[1].x[j] = pp[1].x[i1]; pp[1].x[i1] = i;
}
if (fd >= 0)
    (void) close(fd);
if (fj) {
60     (void) unlink(jname);
    fj = 0;
    offset = 0;}}

```

FIGURE 4Q

```
/*
 * write a filled jmp struct offset of the prev one (if any): nw()
 */
writejumps(ix)
{
    int      ix;
    char      *mktemp();

    if (!fj) {
        if (mktemp(jname) < 0) {
            fprintf(stderr, "%s: can't mktemp() %s\n", prog, jname);
            cleanup(1);
        }
        if ((fj = fopen(jname, "w")) == 0) {
            fprintf(stderr, "%s: can't write %s\n", prog, jname);
            exit(1);
        }
    }
    (void) fwrite((char *)&dx[ix] jp, sizeof(struct jmp), 1, fj),
    (void) fwrite((char *)&dx[ix] offset, sizeof(dx[ix] offset), 1, fj);
}
```

writejumps

5
10
15
20
25
30
35
40
45
50
55
60

FIGURE 5

[illegible]

FIGURE 6

5 CACAACCAGCCACCCCTCTAGGATCCCAGCCCAGCTGGTGCTGGGCTCAGAGGAGAAGGC
 CCCGTGTTGGGAGCACCCCTGCTTGCCTGGAGGGACAAGTTTCCGGGAGAGATCAATAAAG
 GAAAGGAAAGAGACAAGGAAGGGAGAGGTCAGGAGAGCGCTTGATTGGAGGAGAAGGGCC
 AGAGAATGTCGTCCCAGCCAGCAGGGAACCAGACCTCCCCGGGGCCACAGAGGACTACT
 CCTATGGCAGCTGGTACATCGATGAGCCCCAGGGGGGCGAGGAGCTCCAGCCAGAGGGGG
 AAGTGCCCTCCTGCCACACCAGCATAACCACCCGGCCTGTACCACGCCTGCCTGGCCTCGC
 10 TGTCAATCCTTGTGCTGCTGCTCCTGGCCATGCTGGTGAGGCGCCGCCAGCTCTGGCCTG
 ACTGTGTGCGTGGCAGGCCCCGGCCTGCCAGGCCCCGGGCAGTGCCTGCTGCTGTTTTCA
 TGGTCCTCCTGAGCTCCCTGTGTTTGTGCTGCTCCCCGACGAGGACGCATTGCCCTTCCTGA
 CTCTCGCCTCAGCACCCAGCCAAGATGGGAAAAGTGAAGGCTCCAAGAGGGGGCCTGGAAGA
 TACTGGGACTGTTCTATTATGCTGCCCTCTACTACCCTCTGGCTGCCTGTGCCACGGCTG
 15 GCCACACAGCTGCACACCTGCTCGGCAGCACGCTGTCTGGGCCACCTTGGGGTCCAGG
 TCTGGCAGAGGGCAGAGTGTCCCCAGGTGCCCAAGATCTACAAGTACTACTCCCTGCTGG
 CCTCCCTGCCTCTCCTGCTGGGCCTCGGATTCTGAGCCTTTGGTACCCTGTGCAGCTGG
 TGAGAAGCTTCAGCCGTAGGACAGGAGCAGGCTCCAAGGGGCTGCAGAGCAGCTACTCTG
 AGGAATATCTGAGGAACCTCCTTTGCAGGAAGAAGCTGGGAAGCAGCTACCACACCTCCA
 20 AGCATGGCTTCCTGTCCTGGGCCCCGCGTCTGCTTGAGACACTGCATCTACACTCCACAGC
 CAGGATTCCATCTCCCGCTGAAGCTGGTGCTTTCAGCTACACTGACAGGGACGGCCATTT
 ACCAGGTGGCCCTGCTGCTGCTGGTGGGCGTGGTACCCACTATCCAGAAGGTGAGGGCAG
 GGGTCACCACGGATGTCTCCTACCTGCTGGCCGGCTTTGGAATCGTGCTCTCCGAGGACA
 AGCAGGAGGTGGTGGAGCTGGTGAAGCACCATCTGTGGGCTCTGGAAGTGTGCTACATCT
 25 CAGCCTTGGTCTTGTCTGCTTACTCACCTTCCTGGTCCTGATGCGCTCACTGGTGACAC
 ACAGGACCAACCTTCGAGCTCTGCACCGAGGAGCTGCCCTGGACTTGAGTCCCTTGATC
 GGAGTCCCCATCCCTCCCGCCAAGCCATATTCTGTTGGATGAGCTTCAGTGCCCTACCAGA
 CAGCCTTTATCTGCCTTGGGCTCCTGGTGCAGCAGATCATCTTCTTCTGGGAACACCGG
 CCCTGGCCTTCCTGGTGCTCATGCCTGTGCTCCATGGCAGGAACCTCCTGCTCTTCCGTT
 30 CCCTGGAGTCCTCGTGGCCCTTCTGGCTGACTTTGGCCCTGGCTGTGATCCTGCAGACA
 TGGCAGCCCATTTGGGTCTTCCTGGGAGACTCATGATGGACACCCACAGCTGACCAACCGGC
 GAGTGCTCTATGCAGCCACCTTTCTTCTCTTCCCCCTCAATGTGCTGGTGGGTGCCATAG
 TGGCCACCTGGCGAGTGCTCCTCTCTGCCCTCTACAACGCCATCCACCTTGGCCAGATGG
 ACCTCAGCCTGCTGCCACCGAGAGCCGCCACTCTCGACCCCGGCTACTACACGTACCGAA
 35 ACTTCTTGAAGATTGAAGTCAGCCAGTCGCATCCAGCCATGACAGCCTTCTGCTCCCTGC
 TCCTGCAAGCGCAGAGCCTCCTACCCAGGACCATGGCAGCCCCCAGGACAGCCTCAGAC
 CAGGGGAGGAAGACGAAGGGATGCAGCTGCTACAGACAAAGGACTCCATGGCCAAGGGAG
 CTAGGCCCGGGGCCAGCCGCGGCAGGGCTCGCTGGGGTCTGGCCTACACGCTGCTGCACA
 ACCCAACCCTGCAGGTCTTCCGCAAGACGGCCCTGTTGGGTGCCAATGGTGCCAGCCCT
 40 GAGGGCAGGGAAGGTCAACCCACCTGCCCATCTGTGCTGAGGCATGTTCTGCCTACCAC
 CTCTCCCTCCCCGGCTCTCCTCCCAGCATCACACCAGCCATGCAGCCAGCAGGTCTCTCC
 GGATCACTGTGGTTGGGTGGAGGTCTGTCTGCACTGGGAGCCTCAGGAGGGCTCTGCTCC
 ACCCACTTGGCTATGGGAGAGCCAGCAGGGGTTCTGGAGAAAGAACTGGTGGGTAGGG
 CCTTGGTCCAGGAGCCAGTTGAGCCAGGGCAGCCACATCCAGGCGTCTCCCTACCCTGGC
 45 TCTGCCATCAGCCTTGAAGGGCCTCGATGAAGCCTTCTCTGGAACCACTCCAGCCCAGCT
 CCACCTCAGCCTTGGCCTTCACGCTGTGGAAGCAGCCAAGGCACTTCTCACCCCCCTCAG
 CGCCACGGACCTCTCTGGGGAGTGGCCGGAAAGCTCCCGGGCCTCTGGCCTGCAGGGCAG
 CCAAGTCATGACTCAGACCAGGTCCCACACTGAGCTGCCACACTCGAGAGCCAGATAT
 TTTTGTAGTTTTTATGCCTTTGGCTATTATGAAAGAGGTTAGTGTGTTCCCTGCAATAAA
 50 CTTGTTCCCTGAGAAAAA

FIGURE 7

5 MSSQPAGNQTSPGATEDYSYGSWYIDE PQGGEELQPEGEVPSCHTSIPPGLYHACLASL
SILVLLLLLAMLVRRRQLWPDCVRGRPGLPRPRAVPAAVFMVLLSSLCLLLPDEDALPFL
TLASAPSQDGKTEAPRGAWKILGLFYAAALYYPLAACATAGHTAAHLLGSTLSWAHLGV
QVWQRAECPQVPKIYKYSSLASLPLLLGLGFLSLWYPVQLVRSFSRRTGAGSKGLQSS
YSEEYLRNLLCRKKLGSSYHTSKHGFSLWARVCLRHCIYTPQPGFHLPLKLVLSATLTG
TAIYQVALLLLVGVVPTIQKVRAGVTTDVSYLLAGFGIVLSEDKQEVVELVKHHLWALE
10 VCIYISALVLSCLLTFLVLMRSLVTHRTNLRALHRGAALDLSPLHRSPHPSRQAIFCWMS
FSAYQTAFICLGLLVQQIIFFLGTTALAFVLMPVLHGRNLLLFRSLESSWPFWLTLAL
AVILQNMAAHWVFLETHDGHPQLTNRRVLYAATFLFPLNVLVGAIVATWRVLLSALYN
AIHLGQMDLSLLPPRAATLDPGYTYRNFLKIEVSQSHPAMTAFCSLLLQAQSLLPRTM
AAPQDSL RPGEEDGMQLLQTKDSMAKGARPGASRGRARWGLAYTLLHNPTLQVFRKTA
LLGANGAQP

15 Important features of the protein:

Signal peptide:

none

Transmembrane domain:

54-71

93-111

140-157

197-214

291-312

356-371

425-444

464-481

505-522

Motif name: N-glycosylation site.

8-12

Motif name: N-myristoylation site.

50-56

167-173

232-238

308-314

332-338

516-522

618-624

622-628

631-637

652-658

Motif name: Prokaryotic membrane lipoprotein lipid attachment site.

355-366

Motif name: ATP/GTP-binding site motif A (P-loop).

123-131

Stra6 Variant Clones

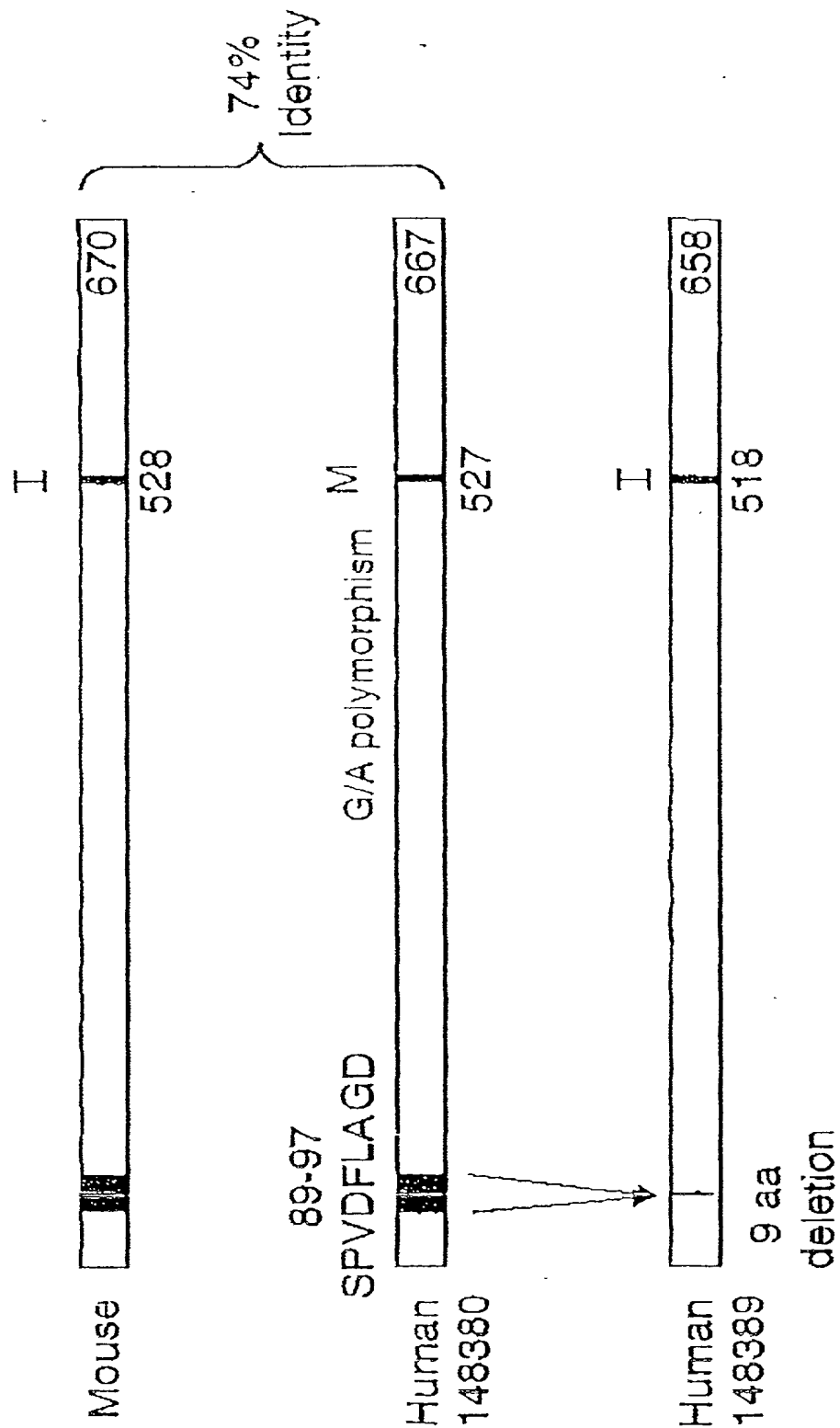


FIGURE 8

Hydrophobicity Plot of Human Stra6

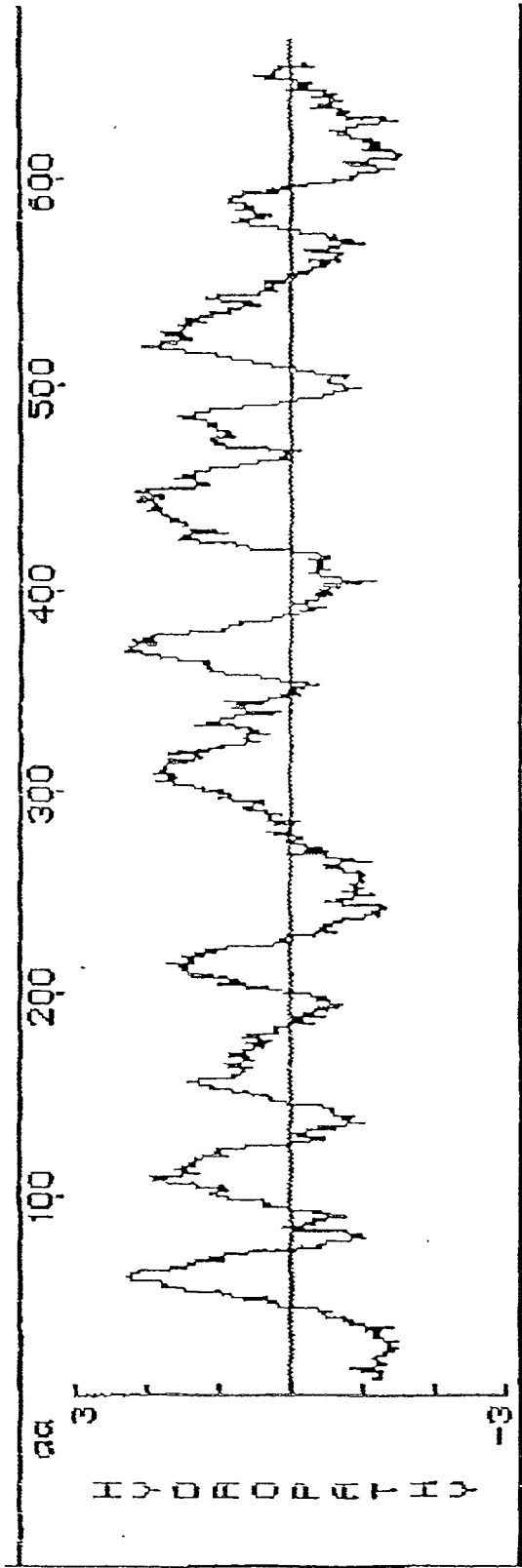
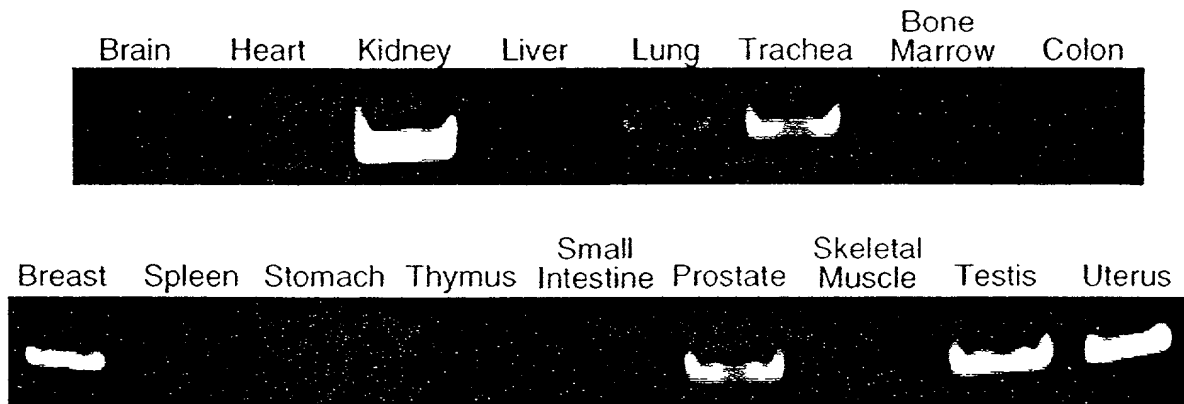


FIGURE 9

- 3 kb mRNA
- 667 Amino Acids --> 50% Residues Hydrophobic
- 73.5 kDa Protein
- 9 Potential Transmembrane Domains

FIGURE 10



Stra6 RNA Expression in Human Colon Tumor Tissue

FIGURE 11

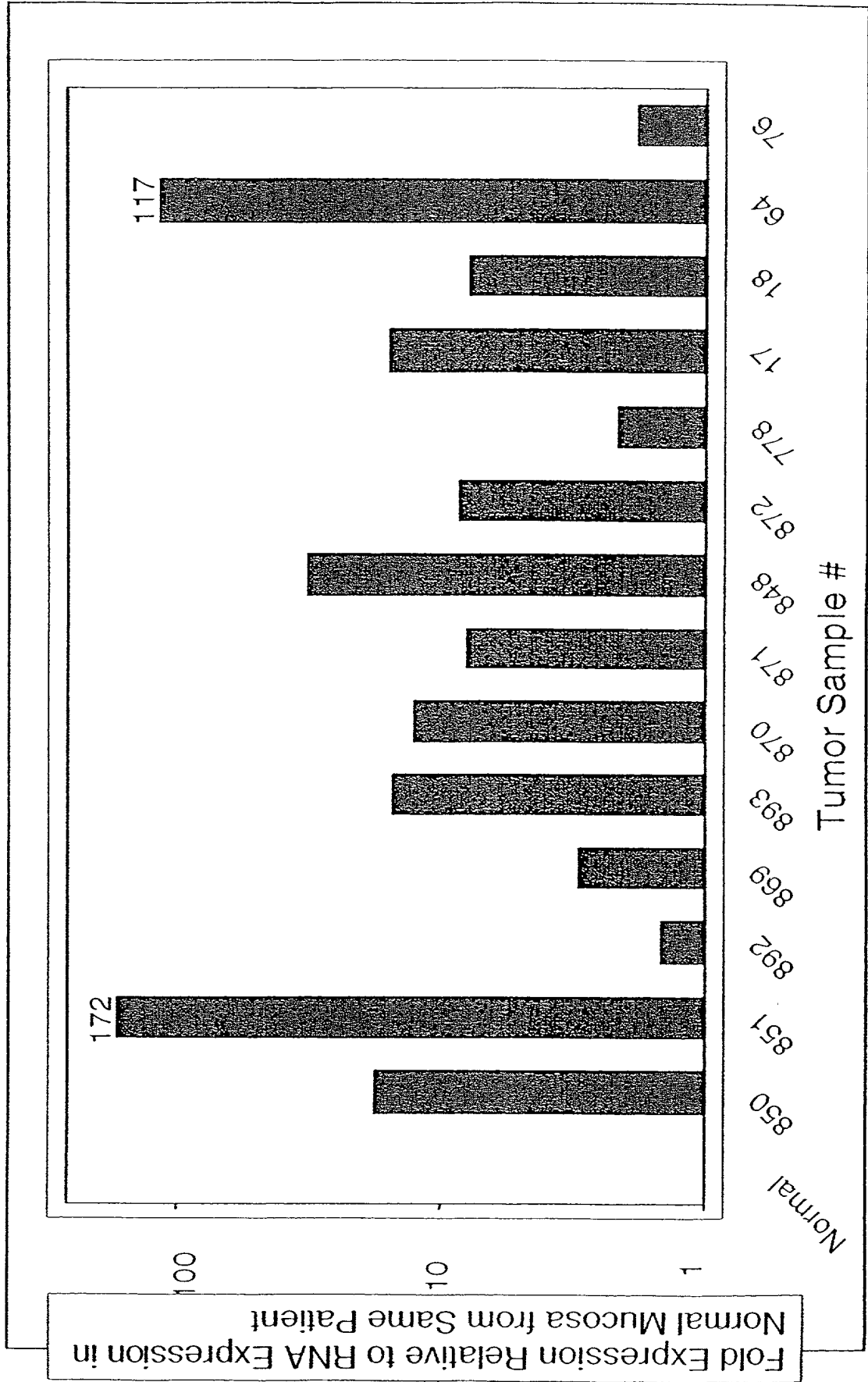
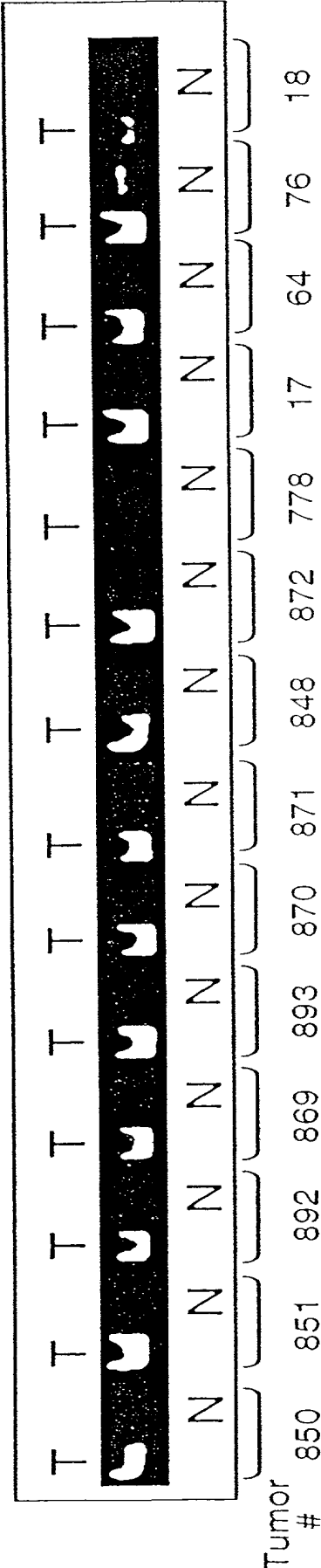


FIGURE 12A

**Stra6 RNA Expression in Human Colon Tumor
Tissue vs Normal Mucosa From the Same Patient**

Taqman Product Analysis After 40 Cycles

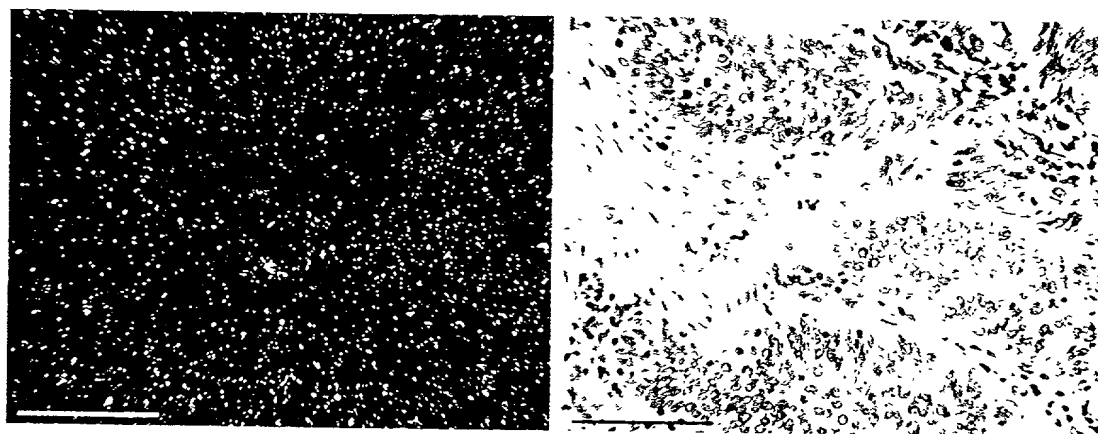
Stra6



GAPDH



FIGURE 12B



C

09904842 094004

FIGURE 13

FIGURE 13

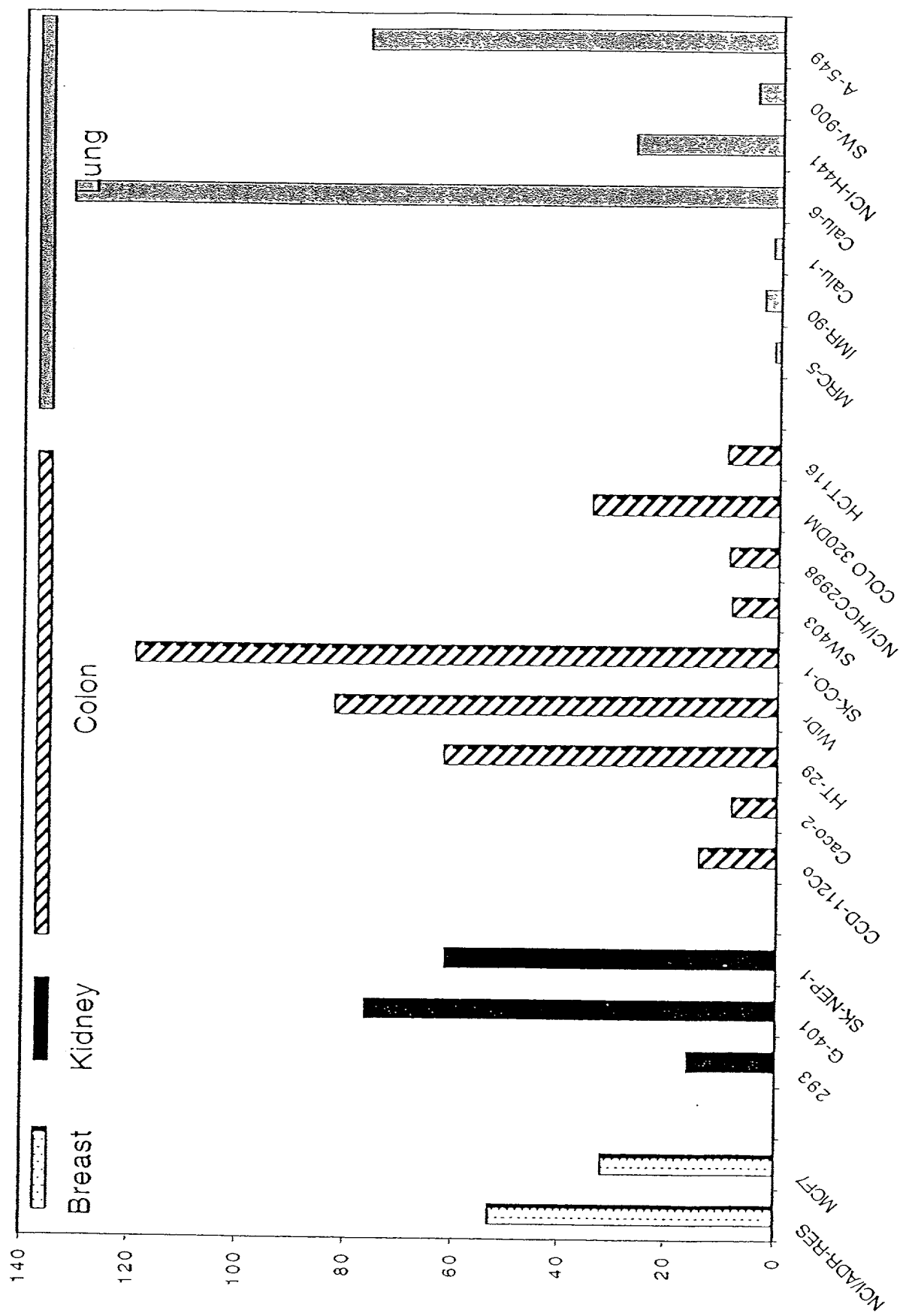
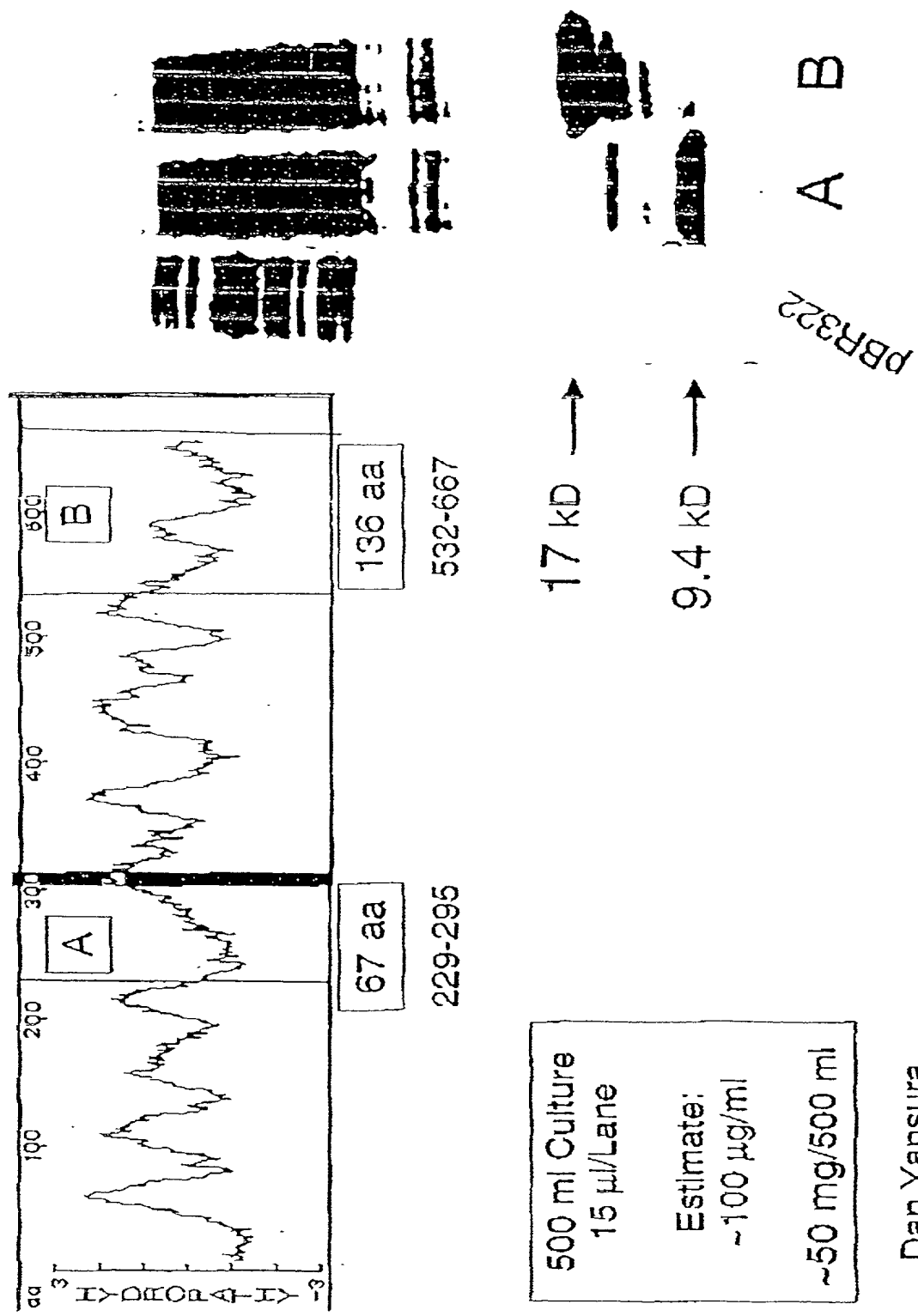


FIGURE 14

Stra6 Peptide Expression in E. coli Poly-His Cleavable Leader at N-Terminus



Dan Yansura

FIGURE 15

Stra6 RNA Expression in Human Colon Carcinoma Cells +/- Retinoic Acid

TM#75 (2/28/00)

VD3 - vitamin D3 (1µM); ATRA - all-trans-retinoic acid (1 µM)

9cRA - 9-cis-retinoic acid (1 µM)

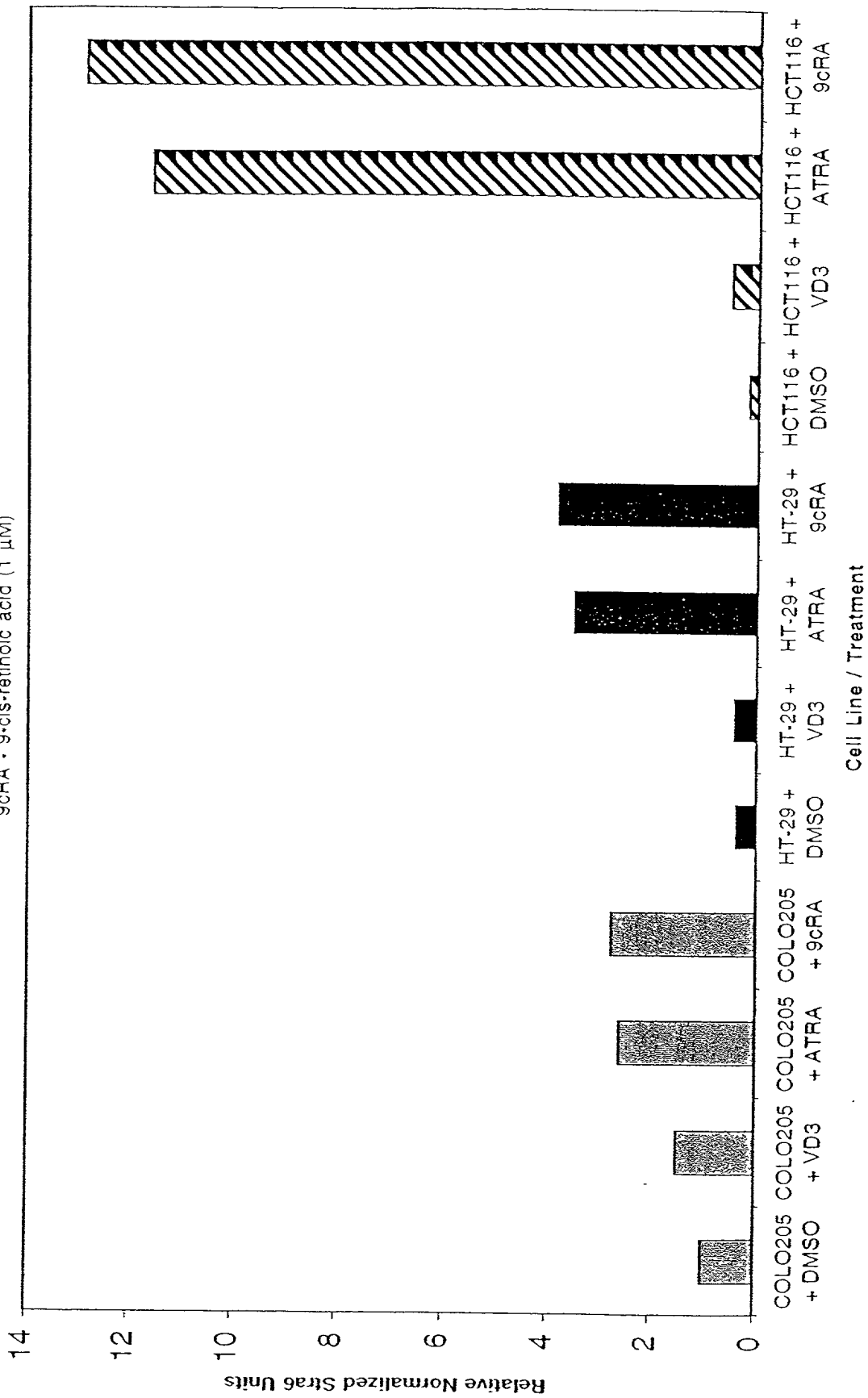
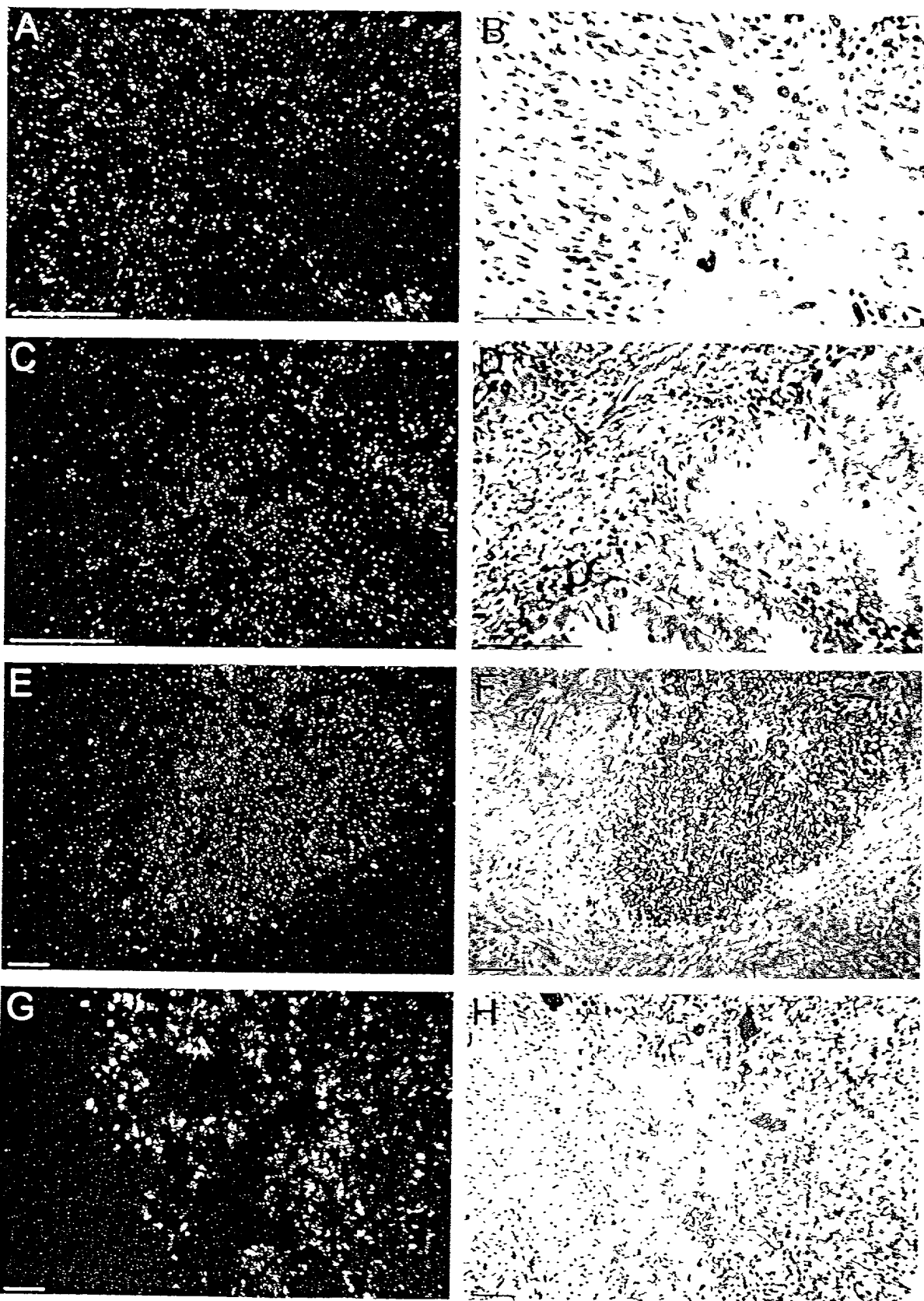


FIGURE 16



09901813-071001

FIGURE 17

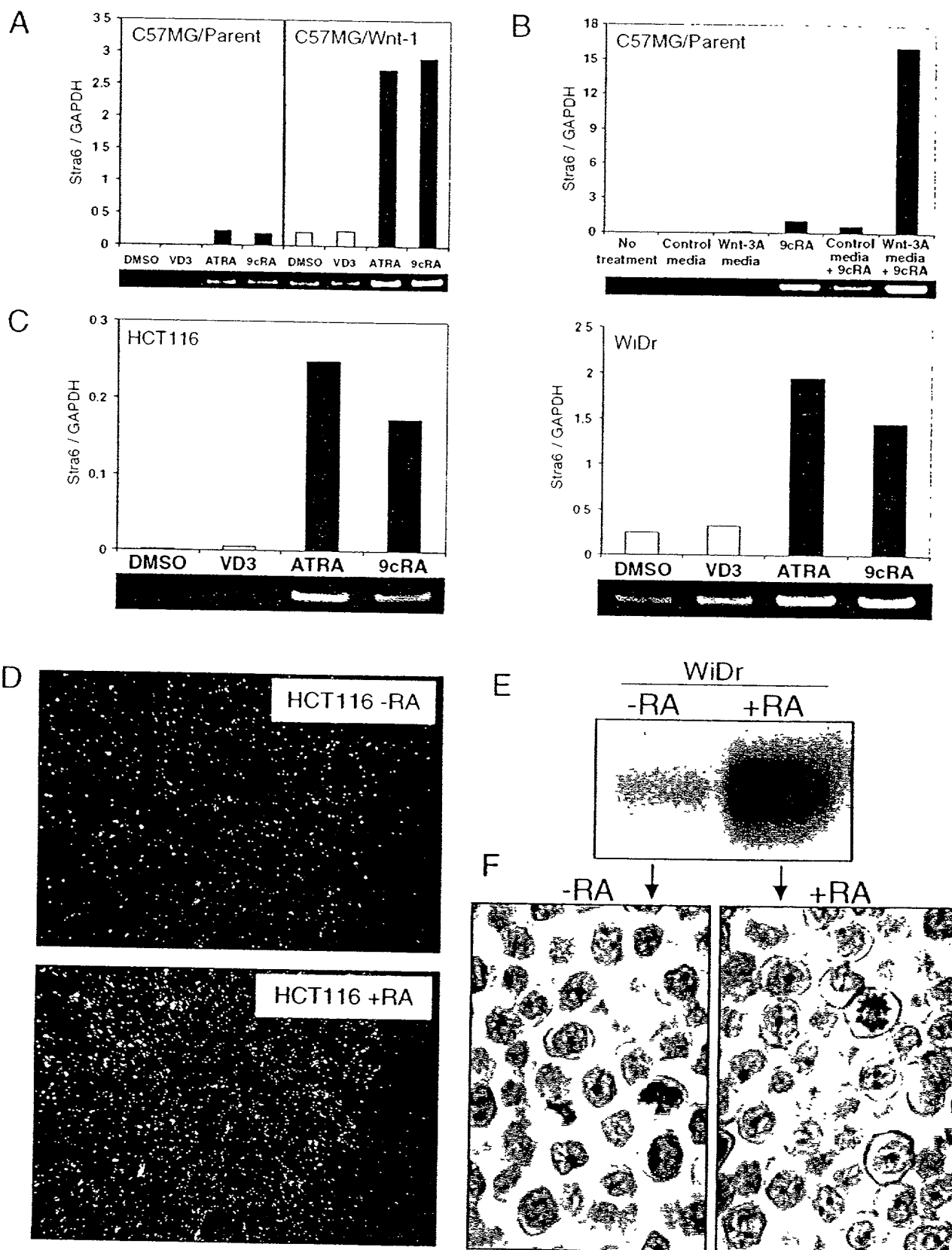
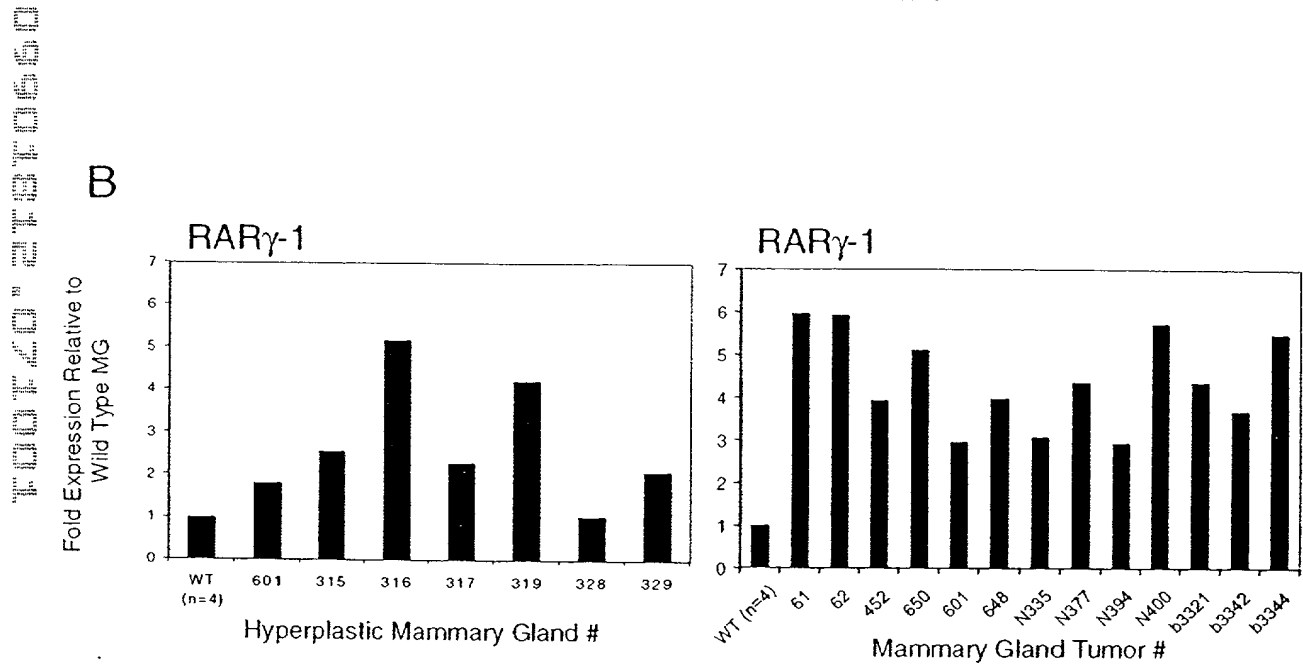
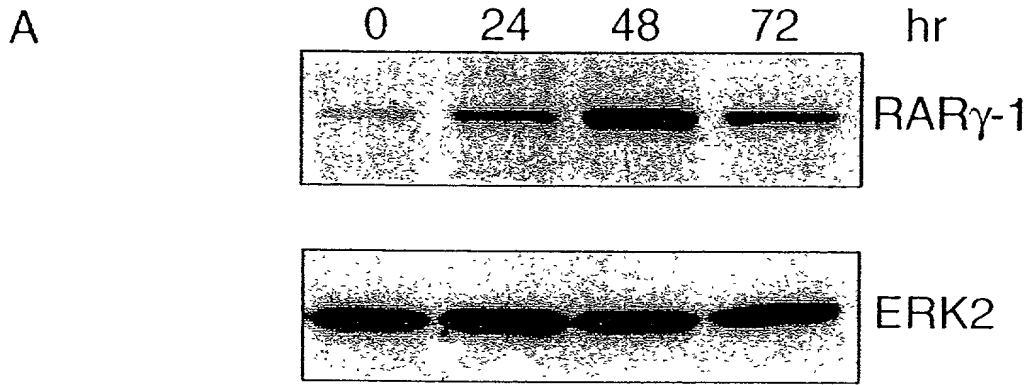


FIGURE 18



Array

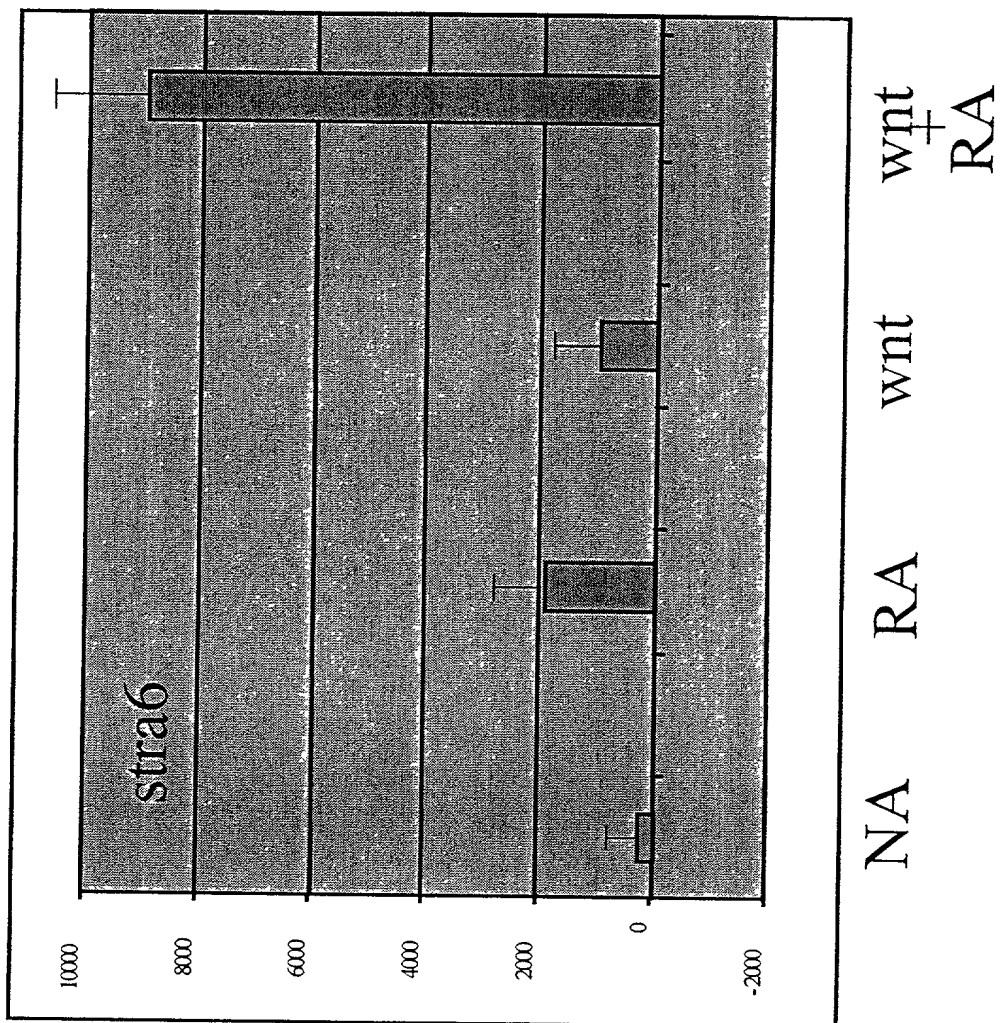
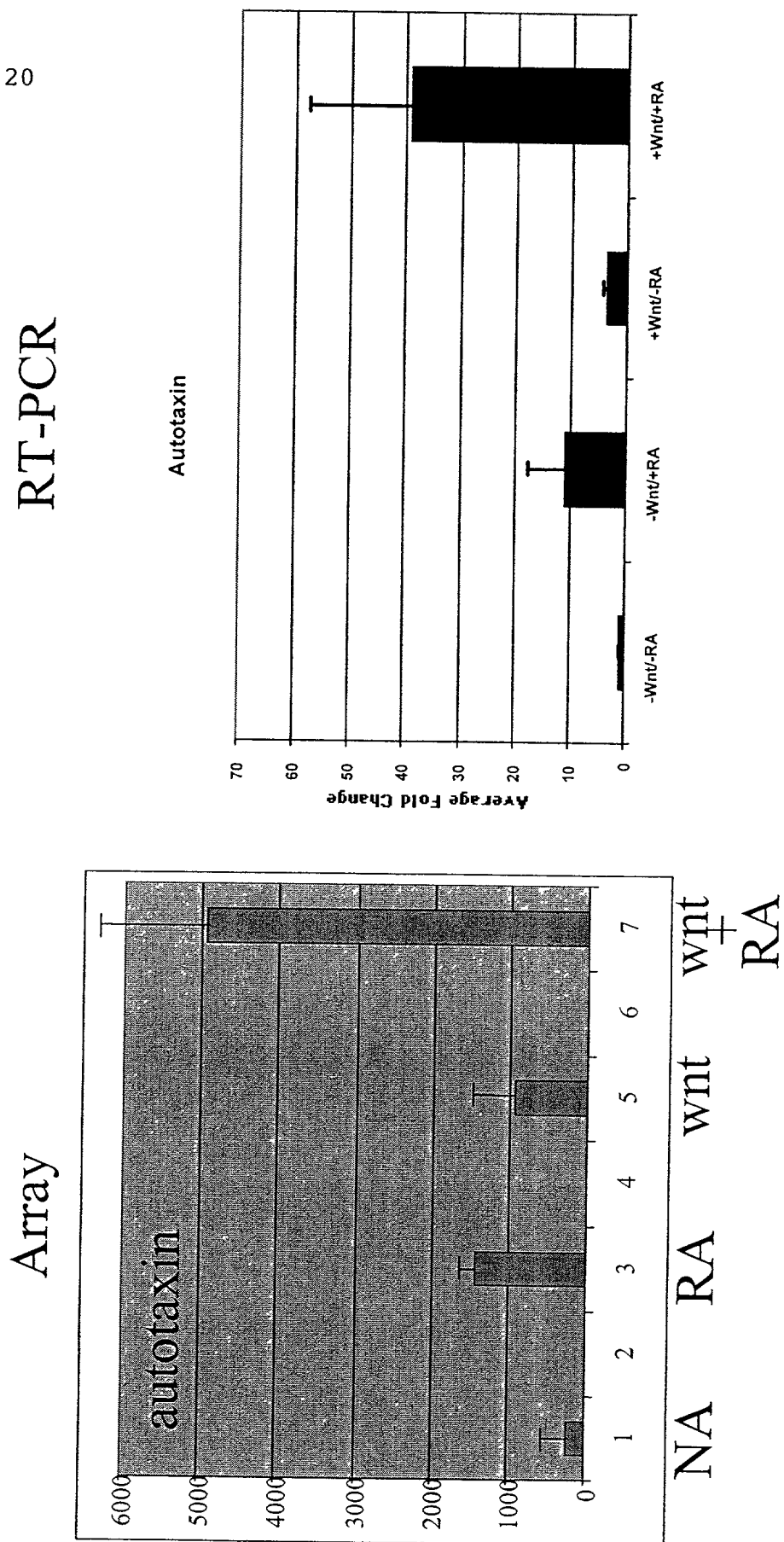
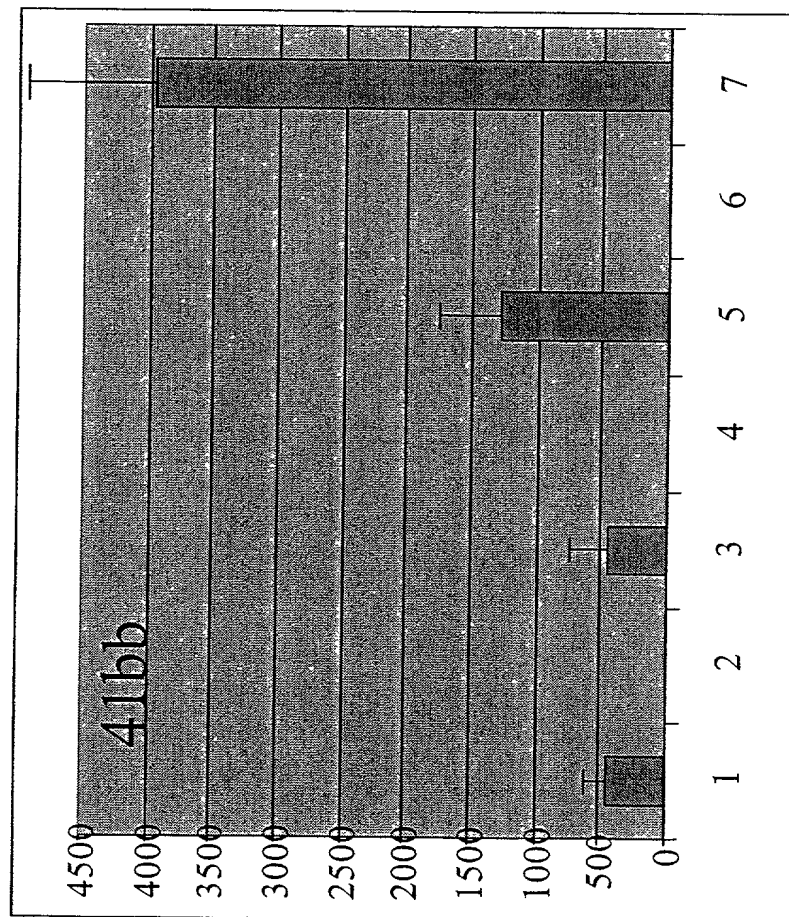


Figure 19

Figure 20



Array



NA RA wnt wnt + RA

RT-PCR

4-1BB Lige

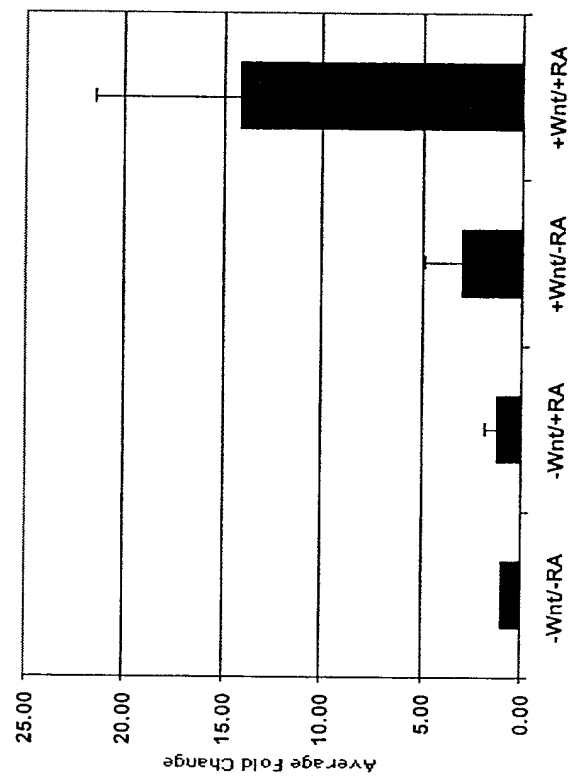
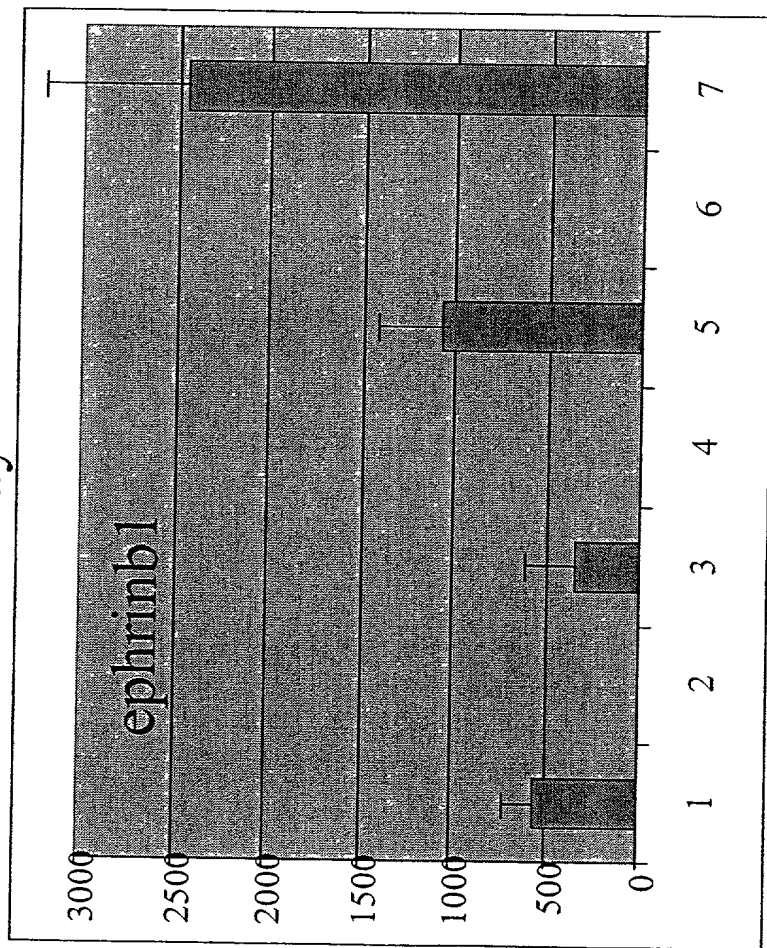


Figure 21

Array



NA RA wnt wnt + RA

RT-PCR

Ephrin

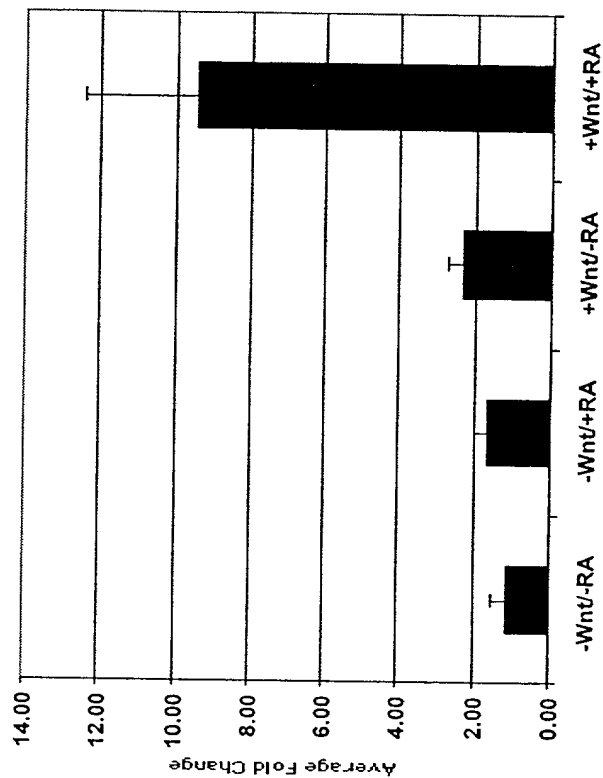
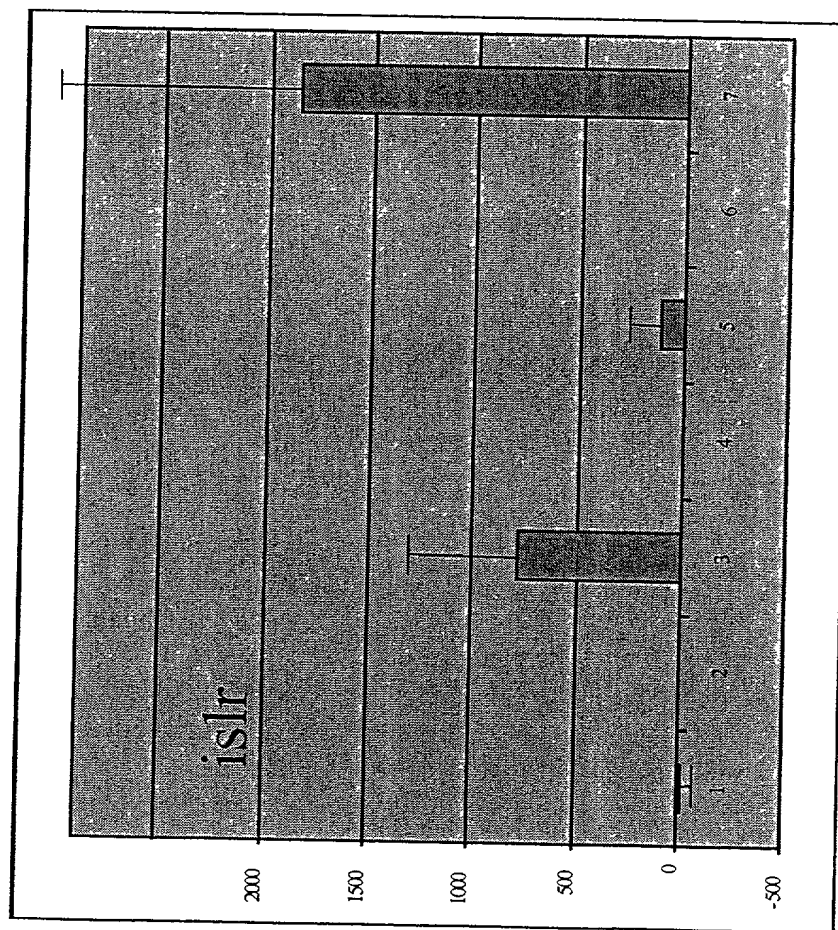


Figure 22

Array



NA RA wnt wnt⁺ RA

RT-PCR

ISL

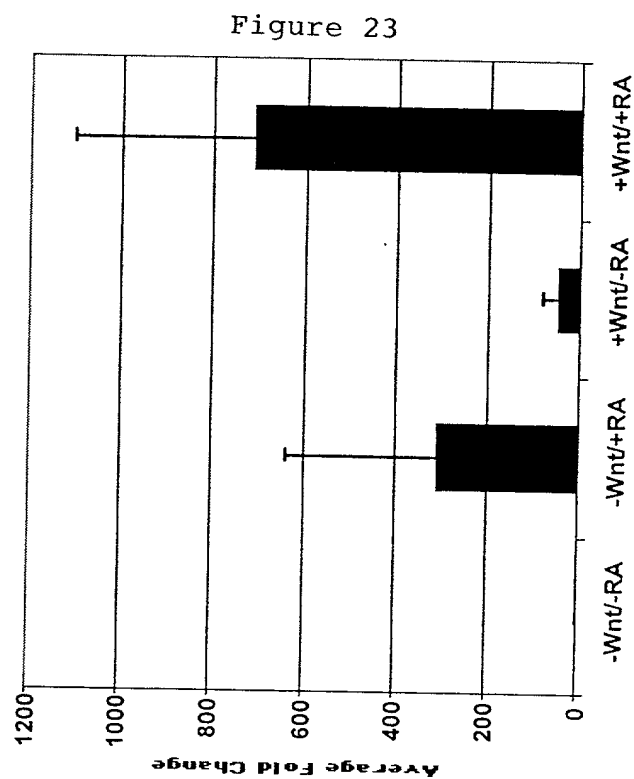


Figure 23

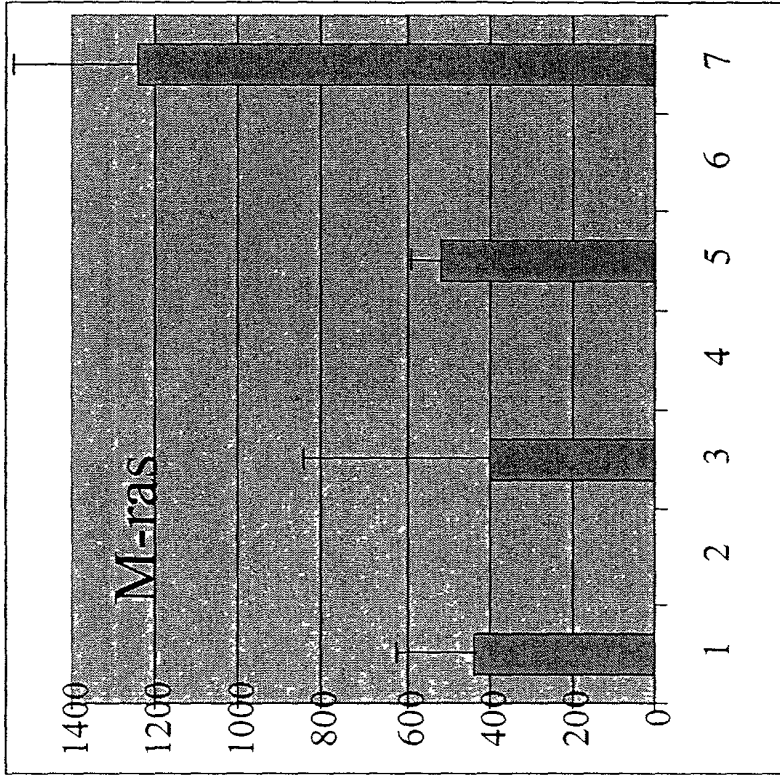
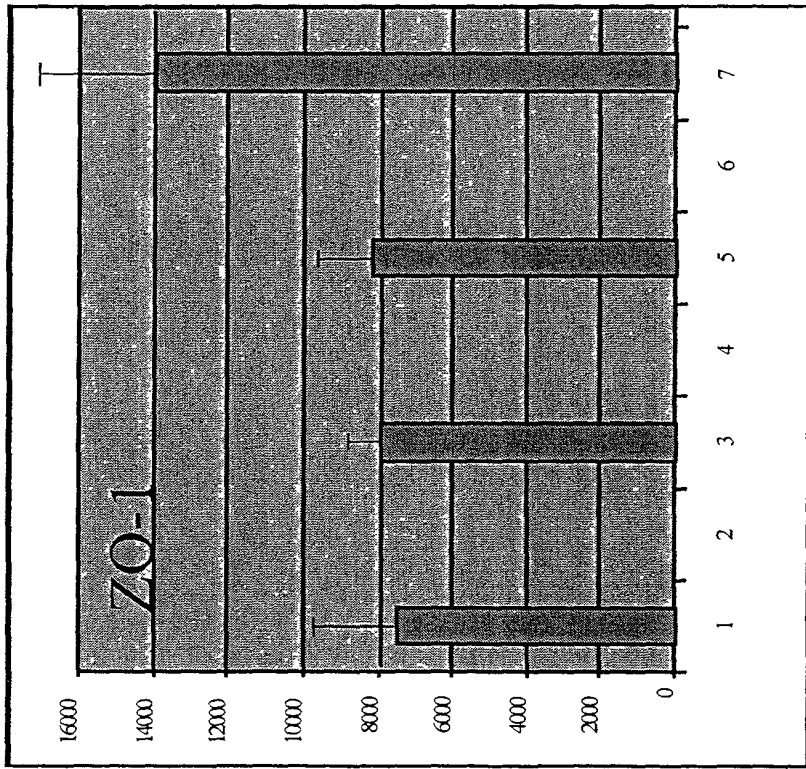


Figure 24

NA RA wnt wnt+ RA

NA RA wnt wnt+ RA

Figure 25

Figure 25 A

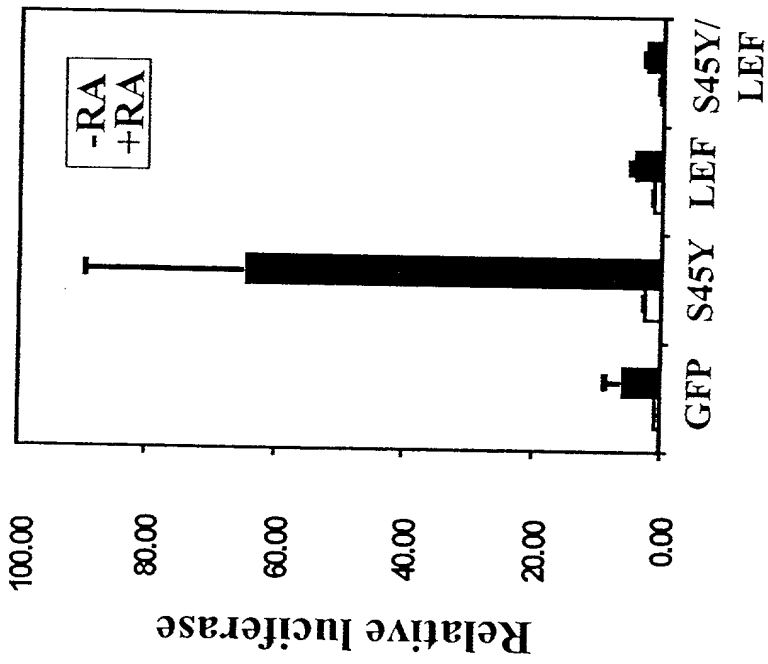


Figure 25 B

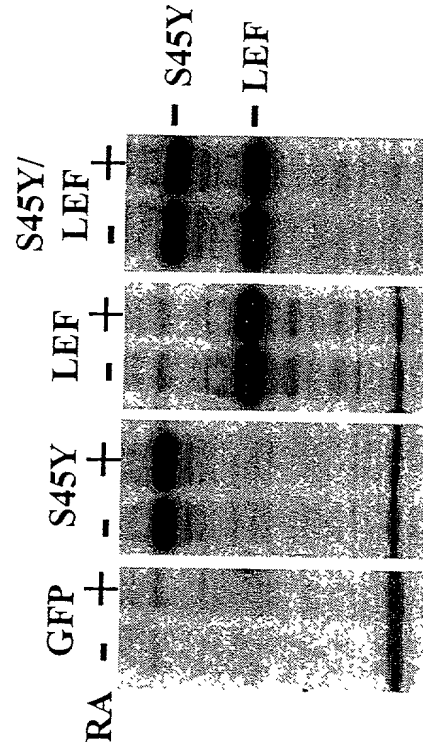
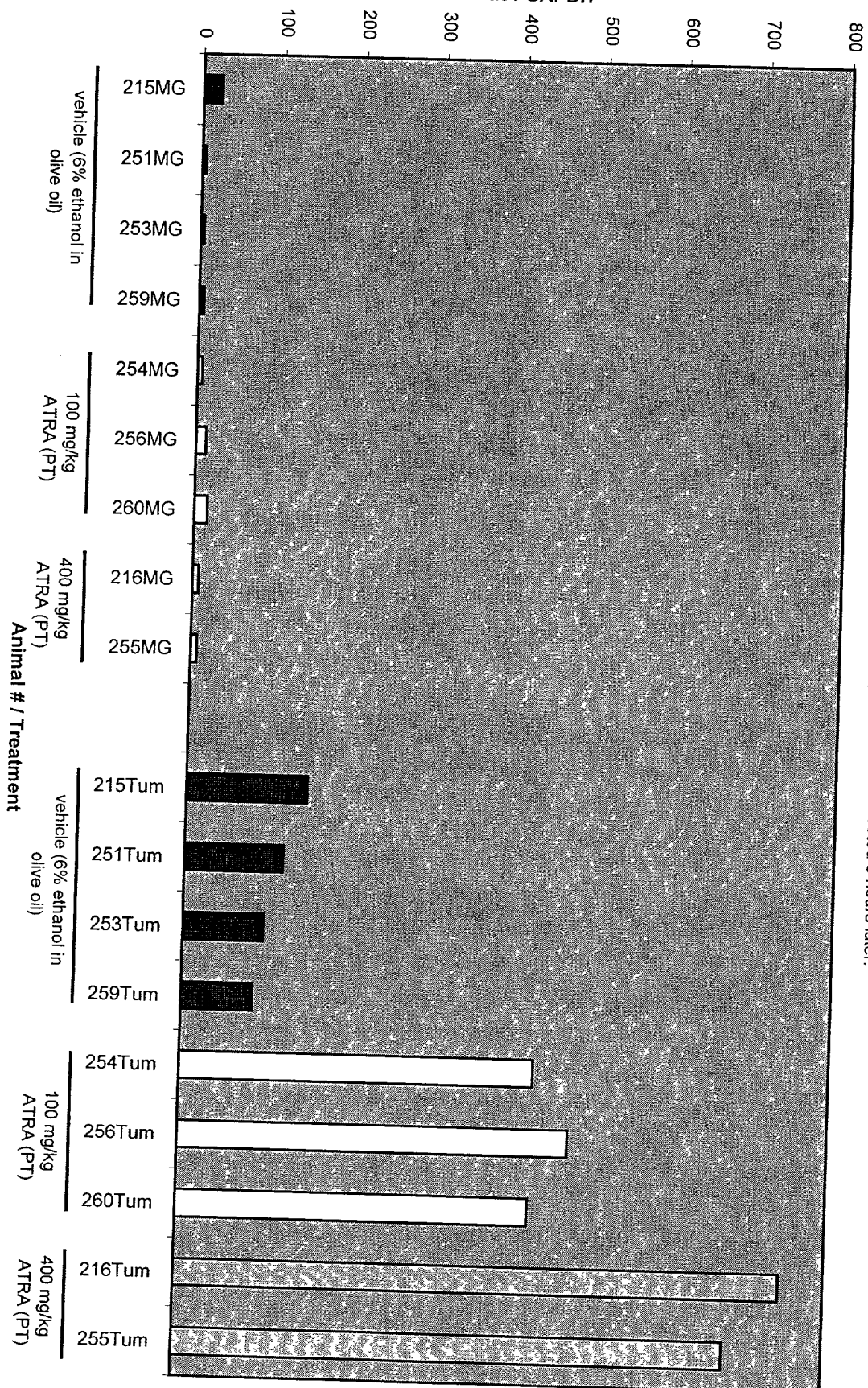


Figure 26
Stra6 / GAPDH

Stra6 mRNA in normal mouse mammary gland and Wnt-1 mammary gland tumors
Nude mice bearing Wnt-1 tumor explants were given ATRA peri-tumorally (PT) at 100 and 400 mg/kg. Tumors and normal adjacent mammary glands were harvested 8 hours later.

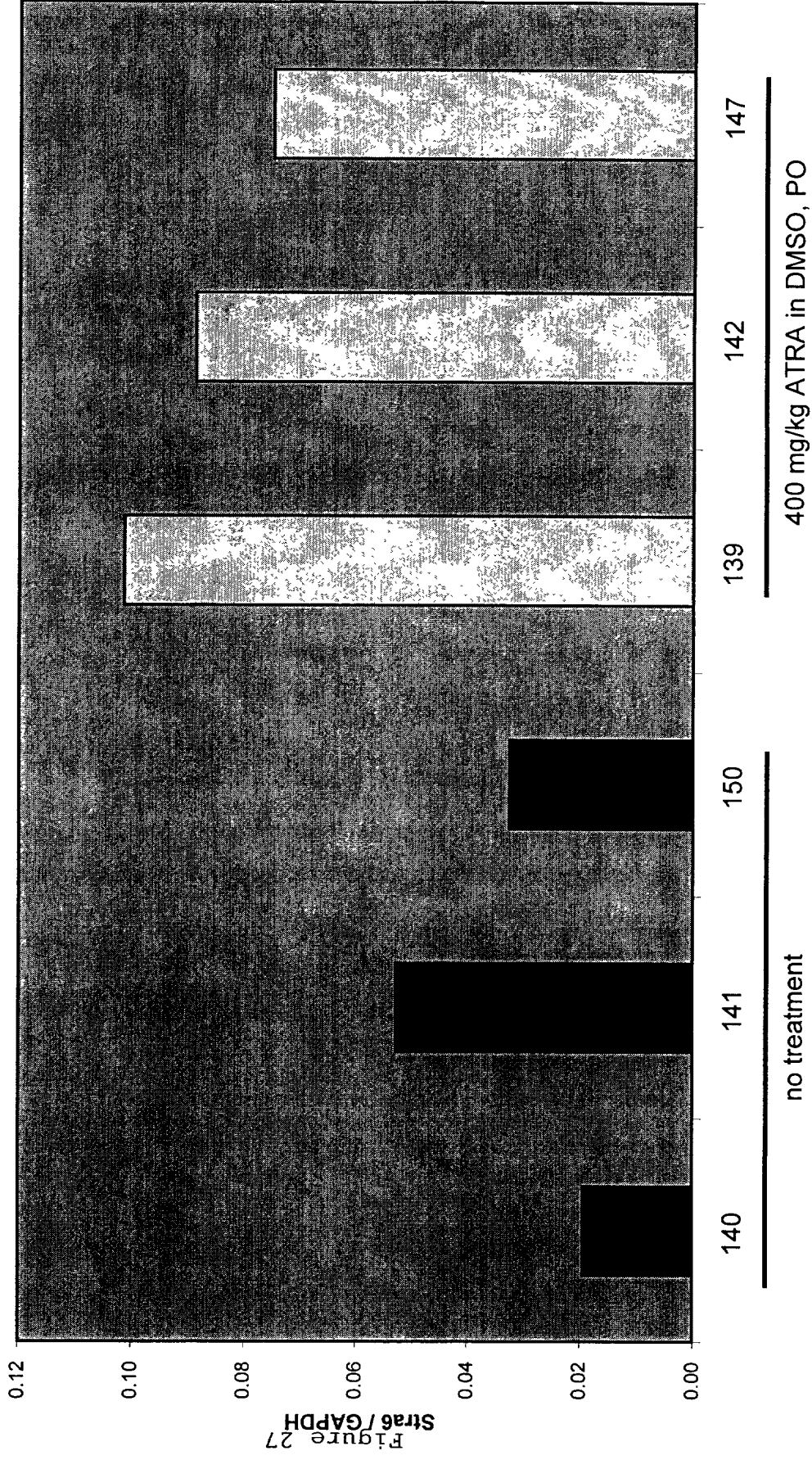


09901813 071001

SCANNED, # 24

Stra6 mRNA in WiDr xenografts from mice dosed with ATRA (400 mg/kg)

Nude mice bearing WiDr xenografts were given ATRA per orum (PO) at 400 mg/kg.
Tumors were harvested 12 hours later.



Animal # / Treatment